

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Motion Control Function Blocks for IEC61131-3

André José Domingues Serra

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Mário Jorge Rodrigues de Sousa (Prof. Dr.)

31 de Julho de 2014

A Dissertação intitulada

“Motion Control Function Blocks for IEC61131-3”

foi aprovada em provas realizadas em 24-07-2014

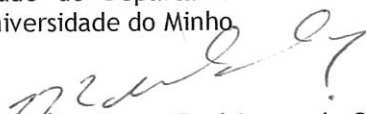
o júri



Presidente Professor Doutor Rui Manuel Esteves Araújo
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto



Professor Doutor Jaime Francisco Cruz Fonseca
Professor Associado do Departamento de Electrónica Industrial da Escola de
Engenharia da Universidade do Minho



Professor Doutor Mário Jorge Rodrigues de Sousa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - André José Domingues Serra



Faculdade de Engenharia da Universidade do Porto

Resumo

A automação industrial é nos dias de hoje uma área em forte crescimento e com uma importância fulcral nas diversas unidades industriais espalhadas pelos diferentes sectores económicos. Tendo em conta a importância dos autómatos usados no ambiente industrial e o facto das linguagens de programação de PLC's definidas na norma IEC 61131-3 não estarem adaptadas para o controlo das máquinas CNC, fez com que o grupo PLCOpen definisse um conjunto de diversos *Function Blocks* que podem ser utilizados na interligação dos PLC's com as máquinas CNC.

Nesta dissertação, são desenvolvidos alguns dos *Function Blocks* criados pela PLCOpen, de forma a serem testados no motor disponível pela Faculdade de Engenharia da Universidade de Porto provando que os FB desenvolvidos pela organização estão aptos a serem utilizados como interligação entre os PLC's e as máquinas CNC. Para o desenvolvimento deste projeto foram usadas ferramentas como a linguagem C de forma a ser produzido um protocolo série que permitisse a comunicação entre o ambiente de desenvolvimento e o motor. Para o desenvolvimento do código presente no interior dos *Function Blocks* foi usado o XML, *eXtensible Markup Language*. O desenvolvimento do projeto foi baseado na norma IEC 61131, nomeadamente na sua terceira parte, esta norma define características mecânicas e lógicas que devem ser seguidas pelos Autómatos Programáveis.

No desenvolvimento do projeto, no âmbito da automação industrial, existem inúmeros ambientes de desenvolvimento lançados pelas diversas marcas de fabricantes de autómatos. Porém estes têm algumas desvantagens, como o facto de serem pagos e de possuírem diversas restrições a nível da plataforma de uso e de compatibilidade. Devido a estes fatores, foi usado o Beremiz, um software criado pela Lolitech. Este ambiente é *open-source*, grátis, multiplataforma e segue as restrições impostas pela norma IEC 61131-3. De modo a validar e avaliar os *Function Blocks* criados, foram realizados inúmeros testes que verificaram se estes comunicavam corretamente com o motor incentivando a sua movimentação.

Abstract

Industrial automation is nowadays an area of strong growth and a central importance in various industrial units spread across different economic sectors. Given the importance of the automatons used in an industrial environment and the fact that the programming languages for PLC's defined in IEC 61131-3 are not suited for the control of CNC machines, has made the PLCOpen group define a number of Function Blocks that can be used in the interconnection of PLCs with CNC machines.

In this dissertation, are developed some of the Function Blocks created by PLCOpen in order to test the engine available from the Faculty of Engineering, University of Porto proving that FB developed by the organization are able to be used as interconnection between the PLC's and CNC machines. For the development of this project tools such as the C language were used in order to be producing a serial protocol allowing communication between the development environment and the engine. For the development of the code within the Function Blocks was used XML, eX-tensible Markup Language. The development of the project was based on the IEC 61131 standard, notably in its third part, this standard defines mechanical characteristics and logic that must be followed by Programmable Logic Controllers.

In the development of the project, within the industrial automation, there are numerous development environments launched by various brands of PLCs. But they have some disadvantages such as the fact that they are paid and they have various restrictions on the level of use platform and compatibility. Due to these factors, we used the Beremiz, software created by Lolitech. This environment is open-source, free, multiplatform and follows the restrictions imposed by IEC 61131-3. In order to validate and evaluate the Function Blocks created, numerous tests that verified if they communicate properly with the engine encouraging their movement were performed.

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Professor Mário Sousa pela disponibilidade e confiança demonstradas, pelo acompanhamento do trabalho e ajuda na resolução de problemas. O meu sincero agradecimento por tudo.

A todos os meus amigos que me acompanharam neste percurso difícil ao longo destes 5 anos, sem vocês não teria chegado onde estou.

Gostaria de agradecer à minha família, aos meus pais por me terem apoiado e proporcionado a oportunidade de frequentar este Mestrado e por toda a educação que me deram ao longo destes 22 anos, ao Bruno, meu irmão mais novo, por ser o meu melhor amigo.

A todos vós, o meu mais sincero obrigado.

André José Domingues Serra

“Believe you can and you’re halfway there.”

Theodore Roosevelt

Conteúdo

1	Introdução	1
1.1	Automação Industrial	1
1.2	Objetivos e Interesse da Dissertação	1
1.3	Estrutura da Dissertação	3
2	Motor Nanotec PD4-N	5
2.1	Visão Geral	5
2.2	Conexões e Configuração	6
2.2.1	Conexões	6
2.2.2	Inputs	8
2.2.3	Outputs	8
2.2.4	Actualização <i>firmware</i>	9
2.2.5	Modos de Operação	10
3	Tecnologias Presentes	13
3.1	Norma IEC 61131-3	13
3.1.1	Visão Geral	13
3.1.2	<i>Program Organization Units</i>	14
3.1.3	Tipos de dados e Variáveis	15
3.1.4	Linguagens de Programação	18
3.2	Beremiz	22
3.2.1	<i>PLC Builder GUI e PLCOpen Editor</i>	22
3.2.2	Compilador Matiec	24
3.2.3	<i>Plugins</i>	25
3.3	RS232	25
3.3.1	Visão Geral	25
3.3.2	Modos de Transmissão	27
3.3.3	Especificação Funcional e Mecânica	27
3.4	Function Blocks for Motion Control	27
3.4.1	Visão Geral	28
3.4.2	Modelo	29
4	Desenvolvimento	59
4.1	Protocolo de Comunicação	59
4.2	Biblioteca MCFunctionBlocks	65
4.3	MC_Function Blocks no Beremiz	70

5	Conclusões e Trabalho Futuro	73
5.1	Conclusões	73
5.2	Trabalho Futuro	74
A	Ligações RS232	75
A.1	Ligações DTE-DCE e DTE-DTE	75
A.2	Sinais para os dispositivos DTE e DCE	76
	Referências	79

Lista de Figuras

1.1	Eixo a ser usado para a validação dos <i>Function Blocks</i>	2
2.1	Configuração dos conectores do motor Nanotec PD4-N [1]	6
2.2	Diagrama de conexões do motor Nanotec PD4-N [1]	7
2.3	Circuito das entradas digitais [1]	8
2.4	Circuito das saídas [1]	9
3.1	Estrutura comum dos 3 tipos de POU [2]	15
3.2	Excerto de código ST [3]	19
3.3	Excerto de código IL [3]	20
3.4	Excerto de código LD [3]	20
3.5	Excerto de código FBD [3]	21
3.6	Excerto de código SFC [3]	21
3.7	Interface Gráfica do Beremiz	23
3.8	Etapas gerais de compilação e organização do código [4]	24
3.9	Lista dos vários <i>Function Blocks</i> disponíveis [5]	29
3.10	Diagrama de Estados dos <i>Motion Control Function Blocks</i> [5]	30
3.11	Exemplo de FB com o tratamento de erros centralizado [5]	31
3.12	Exemplo de FB com o tratamento de erros descentralizado [5]	32
3.13	MC_Power FB [5]	33
3.14	MC_Home FB [5]	34
3.15	MC_Stop FB [5]	35
3.16	MC_Halt FB [5]	36
3.17	MC_MoveAbsolute FB [5]	37
3.18	MC_Relative FB [5]	38
3.19	MC_MoveSuperimposed FB [5]	39
3.20	MC_HaltSuperimposed FB [5]	41
3.21	MC_HaltSuperimposed FB [5]	42
3.22	MC_MoveContinuousAbsolute FB [5]	43
3.23	MC_MoveContinuousRelative FB [5]	44
3.24	MC_TorqueControl FB [5]	46
3.25	MC_PositionProfile FB [5]	47
3.26	MC_VelocityProfile FB [5]	48
3.27	MC_AccelerationProfile FB [5]	49
3.28	MC_SetPosition FB [5]	51
3.29	MC_ReadActualPosition FB [5]	52
3.30	MC_ReadActualVelocity FB [5]	53
3.31	MC_ReadActualTorque FB [5]	54

3.32 MC_Reset FB [5]	55
3.33 MC_DigitalCamSwitch FB [5]	56
4.1 Movement Tab do NanoPro [1]	60
4.2 Ficheiro XML	65
4.3 Exemplo do tipo de dados AXIS_REF	66
4.4 Estrutura AXIS_REF	66
4.5 Declaração das entradas dos FB	67
4.6 Associação das entradas do MC_MoveRelative	67
4.7 Abertura do canal de comunicação	68
4.8 Funções usadas no MC_MoveRelative para comando do motor	68
4.9 Extension.py	69
4.10 MC_MoveRelative no Beremiz	70
4.11 MC_MoveAbsolute no Beremiz	71
4.12 Interligação do FB MC_MoveAbsolute com o FB MC_MoveRelative	72
A.1 Ligação DTE-DCE [6]	75
A.2 Ligação DTE-DTE [6]	76
A.3 Conetor de um dispositivo DTE	76
A.4 Conetor de um dispositivo DCE	77

Lista de Tabelas

2.1	Possíveis estados das saídas [1]	9
2.2	Actualização do <i>firmware</i> [1]	10
3.1	Tipos de dados segundo a norma IEC 61131-3 [7]	16
3.2	Tipos de variáveis segundo a norma IEC 61131-3 [7]	17
3.3	Atributos (qualificadores) das variáveis [7]	18
3.4	Prefixos para a localização e tamanho das <i>directly represented variables</i> [7] . . .	18
4.1	Uso do <i>software</i> NanoPro [1]	60

Abreviaturas e Símbolos

FB	<i>Function Blocks</i>
PLC	<i>Programmable Logic Controller</i>
POU	<i>Program Organization Unit</i>
PROG	<i>Program</i>
FB	<i>Function Block</i>
FUN	<i>Function</i>
ST	<i>Structured Text</i>
IL	<i>Instruction List</i>
LD	<i>Ladder Diagram</i>
FBD	<i>Function Block Diagram</i>
SFC	<i>Sequential Function Chart</i>
EIA	<i>Electronic Industries Association</i>
DTE	<i>Data Terminal Equipment</i>
DCE	<i>Data Circuit-terminating Equipment</i>

Capítulo 1

Introdução

O trabalho desenvolvido nesta Dissertação tem como objetivo a construção de um conjunto de *Function Blocks* de modo a que seja possível controlar um motor da forma pretendida.

1.1 Automação Industrial

O termo automação pode ser definido como um processo onde são realizadas diversas operações industriais com o auxílio de diversos dispositivos eletrónicos e/ou mecânicos que controlam os seus processos.

Com o decorrer dos anos, os sistemas de automação foram evoluindo, deixando de ser sistemas de controlo automático e passando a ser sistemas baseados nas atuais tecnologias [8]. Todas estas mudanças advêm da necessidade de aumentar a produtividade, bem como garantir a melhor relação custo/benefício.

A automação surge como a aplicação de uma variedade de técnicas que juntamente com vários *softwares* e equipamentos específicos de uma determinada máquina executam o processo industrial. São sistemas automatizados que possuem elevados níveis de precisão e sincronismo [9].

Com o grande crescimento da automação industrial, a sua importância tem vindo a aumentar, com a sua introdução em todos os sectores esta fez com que os responsáveis por estes sistemas se deparem com inúmeros desafios, tais como, segurança, confiança nos sistemas, comunicações seguras e a identificação de erros [8].

1.2 Objetivos e Interesse da Dissertação

Este tema está inserido no contexto do uso das várias linguagens da norma IEC 61131-3 para a programação de PLC's (autómatos) em ambientes industriais. A norma IEC 61131-3, é a norma *standard* para o mundo industrial, nomeadamente com o referido anteriormente no campo dos autómatos programáveis.

No entanto, este *software* não está adaptado para o controlo de máquinas CNC. Basicamente, as máquinas CNC são equipamentos que usam um controlador numérico que permite o controlo

das mesmas e é utilizado principalmente em tornos, fresas e robots. O CNC permite o controlo simultâneo de vários eixos, através de um conjunto de instruções escritas num código específico, comandam o movimento do eixo.

Para isso, o grupo PLCOpen definiu os *Function Blocks for Motion Control*, que poderão ser utilizados na interligação de PLC's com as máquinas CNC, isto é, para que estas possam ser programadas usando as linguagens de programação normalizadas.

Numa altura em que os fabricantes de autómatos tentam o melhor possível seguir a norma, este projeto não vai ser exceção. Assim, o estudo da norma IEC 61131, mais em particular a sua terceira parte (IEC 61131-3), que nos descreve as várias linguagens de programação que podem ser usadas bem como conceitos abrangentes e guidelines para o desenvolvimento de projetos para PLC, é um fator importante para o desenvolvimento desta dissertação. Isto é interessante devido ao facto da uniformização e normalização da forma de programar os PLC's com as máquinas CNC usando as linguagens normalizadas pelo IEC 61131-3, para que qualquer pessoa que entenda as linguagens de programação de PLC's os consiga programar para que estes façam a sua função e o código produzido possa ser reutilizado por uma máquina do mesmo tipo mas de outro fabricante, ou seja, haver portabilidade dos programas desenvolvidos.

Para tal, nesta dissertação vão ser implementados alguns dos *Function Blocks* definidos pela PLCOpen, que estão disponíveis no seu website para download (http://www.plcopen.org/pages/tc2_motion_control/downloads/).

Quanto ao ambiente de desenvolvimento, existem no mercado inúmeras escolhas mas todas elas proprietárias e com custo elevado. Neste projeto foi imposto o uso do ambiente Beremiz para o desenvolvimento do projeto.

A validação dos *Function Blocks* desenvolvidos será feita com recurso ao eixo/motor localizado no laboratório I105 e representado na figura 1.1.

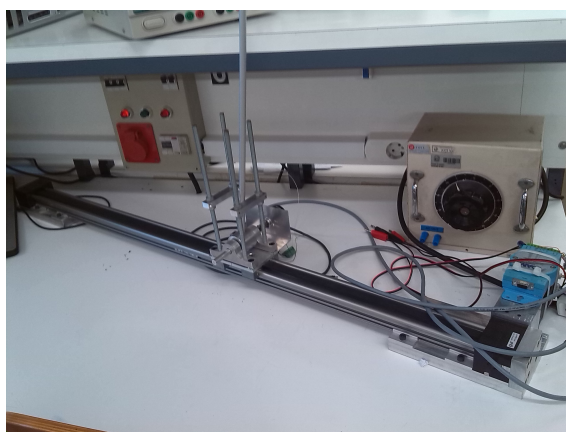


Figura 1.1: Eixo a ser usado para a validação dos *Function Blocks*

De seguida apresenta-se os objetivos na forma de uma lista para que haja uma melhor perceção dos mesmos:

- Desenvolvimento de alguns Function Blocks da PLCOpen;
- Desenvolvimento da aplicação de acordo com a norma no Beremiz;
- Teste e validação da aplicação desenvolvida.

1.3 Estrutura da Dissertação

A Dissertação está dividida em cinco capítulos, constituindo o Capítulo 1, uma contextualização ao tema base do projeto, cuja descrição pormenorizada é apresentada nos capítulos seguintes.

No Capítulo 2 é descrito de forma detalhada o motor Nanotec PD4-N que serviu de validação do trabalho realizado, falando um pouco das suas ligações, *Inputs* e *Outputs* e a sobre a sua configuração e modos de operação.

No Capítulo 3, denominado "Tecnologias Presentes", são descritas todas as tecnologias que foram utilizadas na conceção e desenvolvimento do trabalho. Este inicia-se com uma apresentação da norma IEC 61131, de seguida é apresentado o Beremiz, bem como as suas características e razão para a sua escolha. É apresentado o protocolo de comunicação entre o motor e o ambiente de desenvolvimento, protocolo RS232 e por último são apresentados os *Motion Control Function Blocks*, mostrando quais os FB selecionados e a sua constituição.

No Capítulo 4, é descrito o desenvolvimento do projeto desde o seu início até à última fase de testes e validação.

O Capítulo 5 encerra a Dissertação apresentando as principais conclusões do trabalho desenvolvido. Mais, são propostos possíveis progressos e trabalhos futuros em consequência do trabalho desenvolvido.

Capítulo 2

Motor Nanotec PD4-N

Neste capítulo vai ser apresentado o motor presente no eixo que foi usado para o teste dos Function Blocks e do protocolo desenvolvido, falando-se um pouco do mesmo, das suas características e de como são enviados os comandos para o seu controlo.

2.1 Visão Geral

O motor Nanotec PD4-N é um motor *Plug & Drive* que inclui, para além de uma fase terminal de potência, possui também uma completa rede capaz de controlar a velocidade e posição num *closed loop*, isto é, controlar a velocidade e a posição numa rede em malha fechada [1].

O PD4-N não reduz apenas as despesas de desenvolvimento e de instalação, mas também as necessidades relativas ao espaço e componentes. Este tipo de motor também aumenta a flexibilidade, as propriedades do sistema e a eficácia de uma estação de acionamento. Devido à compatibilidade elétrica e mecânica deste tipo de motores, as atuais soluções podem facilmente ser substituídas por eles [1].

Quanto a como estes motores podem ser operados temos duas hipóteses de firmware viáveis:

- RS485 *firmware*
- CANopen *firmware*

Em relação às funções e características disponibilizadas pelo motor *Plug & Drive* PD4-N estas estão mencionadas na lista seguinte:

- *Microstep* 1/1 – 1/64 estado final do output (0.014° de resolução do step);
- Controlo de corrente em malha fechada, através da comutação de uma onda sinusoidal que provém do *encoder*;
- Poderoso microprocessador DSP para I/O flexíveis;

- Permite a sequência de programas com o NanoJ (RS485), que é uma linguagem integrada baseada no Java, o que significa programas completos que podem ser realizados nos drivers e executados sem um controlador de elevada ordem;
- *Encoder* integrado para monitorização da rotação e do controlo de corrente;
- Interface RS485/CANopen para a parametrização e controlo;
- Uma rede com capacidade para 254 motores (RS485) e 127 motores (CANopen);
- Fácil programação com os programas NanoPro, no caso de ser usado o RS485, ou o Nano-CAN no caso de ser usado o CANopen.

2.2 Conexões e Configuração

Para o correto funcionamento do motor é necessário realizar as ligações necessárias e a instalação do firmware para que este possa aceitar os comandos enviados quer por RS485 ou CANopen. Estas serão abordadas ao longo deste capítulo.

2.2.1 Conexões

Relativamente às conexões de ligação este necessita de três conectores que são os seguintes:

- JST PHD-12: para os *Inputs e Outputs*;
- JST PHD-8: para os protocolos de comunicação RS485 e CANopen;
- Phoenix connector: que serve para a alimentação do motor.

Quanto à configuração destes conectores esta pode ser vista na figura seguinte:

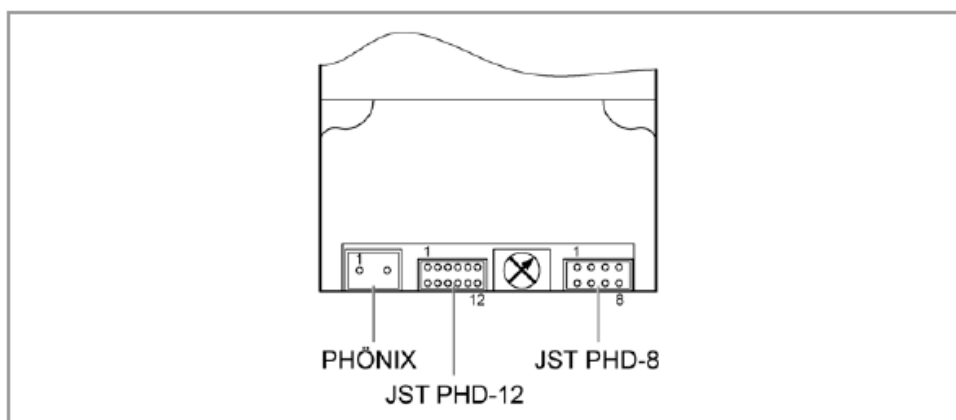


Figura 2.1: Configuração dos conectores do motor Nanotec PD4-N [1]

A alimentação do motor deve estar entre os 12 e os 48 V e não deve ultrapassar os 50 V nem estar abaixo dos 11 V pois se tal acontecer pode danificar as saídas.

Para concluir esta parte, de forma a se poder operar o motor *Plug & Drive* é necessário implementar as ligações de acordo com o diagrama apresentado na figura 2.2:

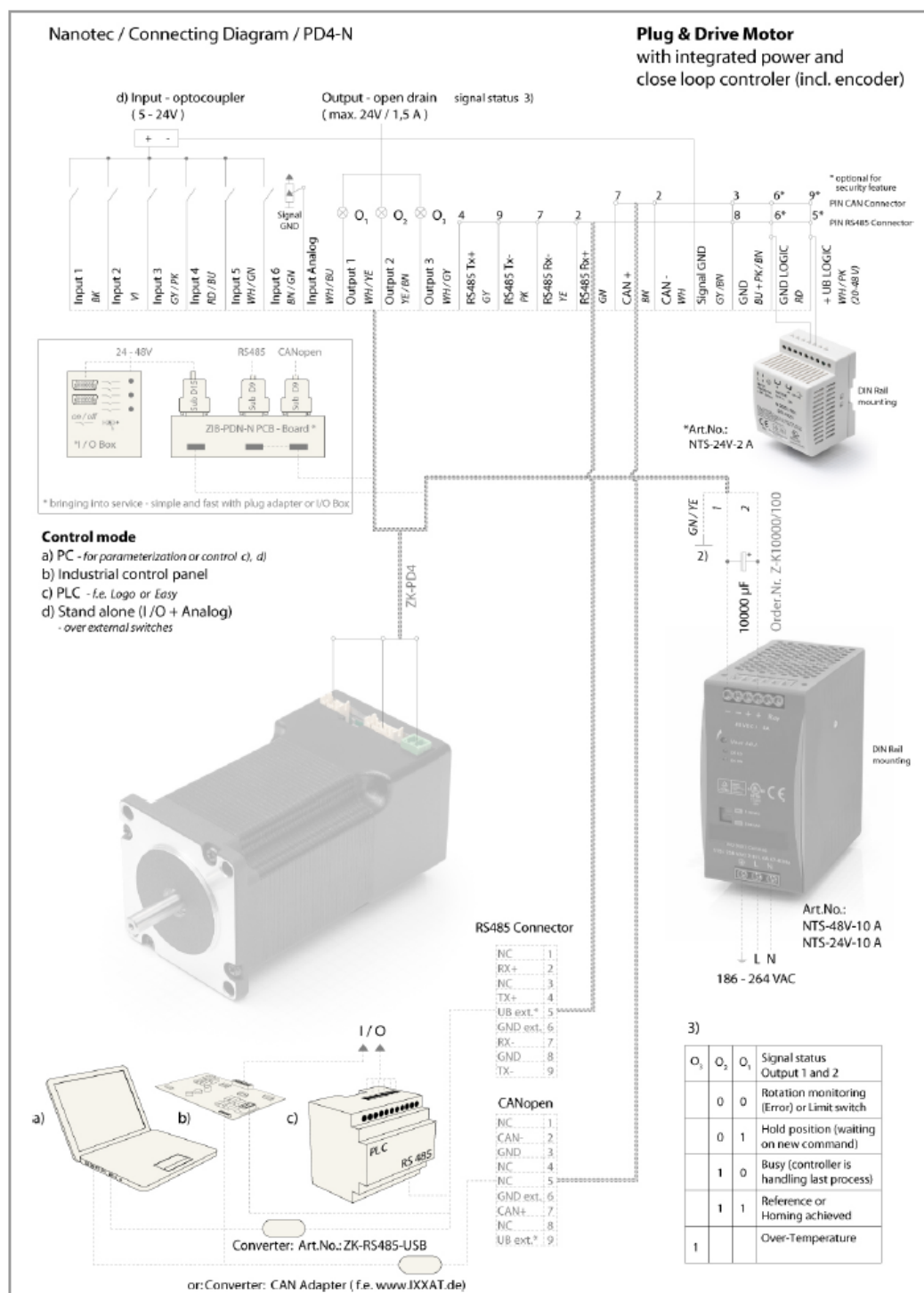


Figura 2.2: Diagrama de conexões do motor Nanotec PD4-N [1]

2.2.2 Inputs

Como se pode ver através da figura anterior este motor possui 6 entradas digitais, 1 entrada analógica e 3 saídas.

Falando um pouco das entradas disponíveis, estas, com exceção da entrada analógica, estão eletricamente isoladas por optoacopladores da fonte de alimentação do PD4-N e projetadas para receberem sinais de entrada com tensão entre os 5-24 V a uma corrente de entrada de 8 mA. É necessário também ter em atenção que a tensão não pode ultrapassar os 24 V.

A figura 2.3 mostra de que forma estão feitas as entradas do motor:

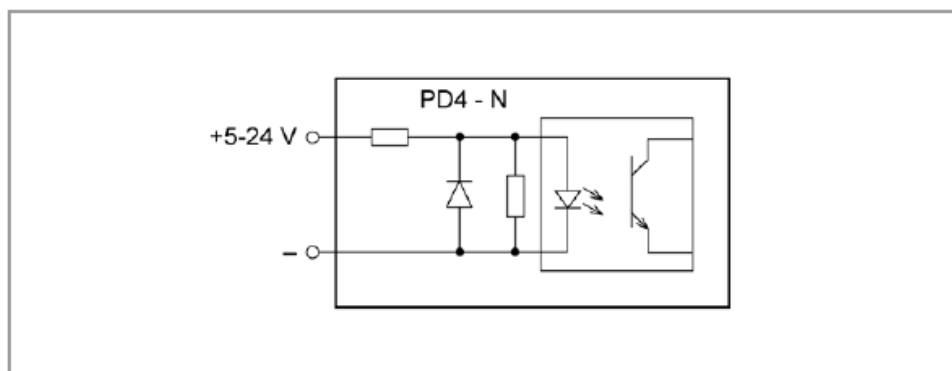


Figura 2.3: Circuito das entradas digitais [1]

Todas as entradas digitais, com exceção do clock no *clock directional mode* podem ser livremente programadas usando o programa NanoPro (RS485) fornecido pela Nanotec, como *switch* para limitar a posição, *enable* entre outras opções e podem ser configuradas como *Active-High* (PNP) ou *Active-Low* (NPN). Podem também ser usadas como controlo sequencial pelo programa NanoJ.

2.2.3 Outputs

Os *Outputs* são saídas a MosFET em topologia *Open-Drain*, ou seja em dreno aberto e que podem aguentar uma tensão até aos 24 V e uma corrente de saída de 2 A.

Para se realizar um teste às saídas pode ser ligado um LED à saída em questão, fazendo com que o LED acenda quando a saída estiver activa.

O circuito das saídas pode ser visto na figura seguinte:

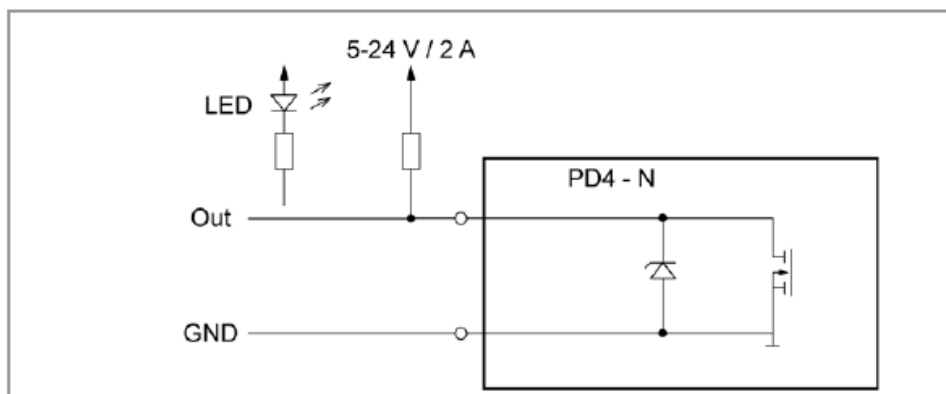


Figura 2.4: Circuito das saídas [1]

O sinal das saídas pode ser visualizado, mas apenas quando é usado o RS485 pois com o CANopen o estado do controlador não é colocado nas saídas.

A Tabela 2.1 mostra os estados possíveis para as 3 saídas disponíveis:

Tabela 2.1: Possíveis estados das saídas [1]

Signal states			Meaning
Output 3	Output 2	Output 1	
	0	0	Rotation Monitoring (error) or limit switch
	0	1	Motor idle (waiting for new command)
	1	0	Busy (control processing last command)
	1	1	Reference point or zero point reached
1			Overtemperature

2.2.4 Actualização *firmware*

Esta secção tem como intuito mostrar como foi configurado o motor e demonstra os primeiros passos a ser realizados para que rapidamente seja possível trabalhar com o motor, se se desejar trabalhar com os programas fornecidos pela Nanotec, o NanoPro (RS485) e o NanoCAN (CANopen). Numa fase inicial do projeto foi usado o NanoPro para verificar se o *hardware* se encontrava nas melhores condições e se a comunicação entre o PC e o PD4-N estava a ocorrer corretamente, este teste vai ser demonstrado mais para frente nesta dissertação.

O PD4-N é sempre entregue com um *firmware* otimizado para o protocolo RS485. Se fosse necessário usar CANopen teria de ser realizado uma actualização de *firmware*.

A Tabela 2.2 mostra como o fazer:

Tabela 2.2: Actualização do *firmware* [1]

Step	Action
1	Install the NanoPro control software on your PC. See the NanoPro separate manual.
2	Connect the PC to the RS485 interface of the Plug & Drive motor according to the connection diagram.
3	In NanoPro open the menu <System / Firmware change / select firmware> and select the desired firmware
4	Click in open

2.2.5 Modos de Operação

Dependendo do percurso a ser efetuado, o motor pode ser colocado em funcionamento usando diferentes modos de operação. Devido à grande capacidade e funções disponíveis, o PD4-N oferece um método mais rápido e simples para a resolução de várias necessidades com menos programação.

O que é necessário fazer é escolher o modo para cada um dos perfis de deslocamento desejados e configurar o controlador de maneira a ir de encontro com os requisitos propostos.

No caso desta dissertação, o NanoPro só foi usado para teste do *hardware*, mas esse teste vai ser exemplificado mais a frente.

De seguida serão apresentados os modos de operação disponíveis quando é usado o protocolo de comunicação série [1]:

- *Relative Positioning* - é um modo usado quando queremos andar uma certa distância em relação a uma dada posição.
- *Absolute Positioning* - modo usado quando se quer que o motor vá da posição A para a posição B.
- *Internal Reference Run* - Durante este modo, o PD4-N viaja para um ponto de referência interno deslocando-se à velocidade mínima escolhida;
- *External Reference Run* - Durante a *External reference run*, o motor viaja até um switch ligado a entrada de referência.
- *Speed Mode* - Este modo deve ser usado quando se deseja fazer o percurso a uma velocidade específica, por exemplo a velocidade de uma bomba. No *Speed Mode*, o motor acelera com uma rampa específica desde de uma velocidade inicial até uma velocidade máxima escolhida.
- *Flag Positioning Mode* - Este modo oferece uma combinação dos modos *Speed* e *Positioning*. Inicialmente, o motor é operado no *Speed Mode*, quando este atinge o ponto escolhido ele muda para o *Positioning Mode* e dirige-se para a posição especificada (relativamente ao ponto em que muda de modo).

- *Clock Direction Mode* - Este modo deve ser usado quando se deseja que o motor opere com um controlador superior, por exemplo, um controlador CNC.

O motor neste modo, é controlado através de 2 entradas com *clock* e a direcção do sinal proveniente de um controlo de posição superior, por exemplo um *indexer*.

Os modos *internal* e *external runs* são suportados neste modo.

- *Analog and Joystick mode* - O motor é controlado simplesmente por um potenciómetro ou um *joystick* (entre -10 V e 10 V).
- *Analogue Positioning mode* - Modo que consiste no deslocamento para uma dada posição. A tensão na entrada analógica é proporcional à posição desejada.
- *Torque Mode* - Este modo deve ser usado quando se quer um torque específico independente da velocidade.

O torque máximo é especificado através da entrada analógica.

No caso de o protocolo escolhido ser o CANopen, apenas 5 modos de operação podem ser escolhidos e são os seguintes [1]:

- *Positioning Mode (PP mode)* - Deve ser usado quando se quer que o motor se dirija de A para B com os parâmetros escolhidos (velocidade, aceleração, etc.).
- *Speed/Velocity Mode* - tal como na comunicação série, é usado quando queremos que o motor se mova com uma velocidade específica.
- *Reference Mode* - Este modo deve ser usado para mandar o motor para a referência escolhida.
- *Interpolated Position Mode* - Este modo deve ser usado com um controlador de caminho.
- *Torque Mode* - Usado para especificar o torque do PD4-N.

Capítulo 3

Tecnologias Presentes

Neste capítulo serão apresentadas as tecnologias incorporadas no projeto. Por um lado a norma IEC 61131, enquanto modelo usado na programação da aplicação de controlo, como também o Beremiz que foi usado como ambiente de desenvolvimento. Por outro lado, protocolo de comunicação RS232 responsável pela comunicação entre a aplicação de controlo e o motor também será alvo de estudo.

3.1 Norma IEC 61131-3

3.1.1 Visão Geral

Desde da criação dos PLC's, muitas das linguagens de programação têm sido utilizadas para escrever programas para máquinas e processos. Antes da criação da norma, como não havia uma forma "universal" de se programar nos PLC's dos vários fabricantes e devido ao constante crescimento da complexidade dos sistemas, houve um grande aumento nos custos e na perda de tempo, pois era necessário o treino de programadores para as aplicações, para a criação de programas maiores e a implementação de cada vez mais sistemas de programação complexos.

Para atenuar esses problemas, a norma IEC 61131 foi criada em 1982 pela mão de um grupo dentro da IEC (*International Electrotechnical Commission*) tendo como objetivo criar uma interface comum ao uso dos autómatos programáveis. Ela está dividida em 8 partes, mas no âmbito desta dissertação, a que tem mais interesse é a terceira parte, pois é esta que aborda o aspeto da programação de controladores industriais e define o modelo de programação e não só. O IEC 61131-3 oferece também conceitos abrangentes e *guidelines* para o desenvolvimento de projetos para PLC [2].

Os principais conceitos e características apresentadas pelo IEC 61131-3 [2] são:

- Declaração de variáveis e alocação estática;
- Tipos de dados;
- Estruturação, modularização, reutilização e portabilidade de *software*;

- Orientação a objetos;
- Processamento multitarefa;
- Cinco linguagens de programação (2 textuais e 3 gráficas).

A norma é considerada como uma *guideline* para a maioria dos fabricantes de PLC's e define vários aspetos para os PLC's, tais como, características do *hardware*, instalação, testes, comunicação e programação, ficando assim estes pontos standardizados para os vários fabricantes e utilizadores desta tecnologia.

Com a norma, tanto os fabricantes como os utilizadores ganham benefícios, como por exemplo, para o caso dos fabricantes esta diminui o risco de estes desenvolverem um sistema que não satisfaça os requisitos do mercado.

Embora a norma defina um conjunto de regras às quais os fabricantes deveriam aderir, tal não acontece de uma forma rígida tendo os fabricantes bastante liberdade em implementar os componentes que pretendem, mesmo quando estes não seguem a norma.

Quanto aos utilizadores, em vez de terem diferentes cursos de programação em PLC's, estes com o IEC 61131-3 apenas precisam de aprender os pontos mais delicados do uso de sistemas de programação individuais e as características especiais dos PLC's que forem adicionadas. Isto "corta" a necessidade do uso de especialistas do sistema e de treino de pessoal e assim os programadores dos PLC's tornam-se mais flexíveis.

Em suma, nos últimos anos e com a crescente necessidade de uniformizar o mercado dos PLC, a aceitação da norma tem aumentado, trazendo várias vantagens aos fabricantes e principalmente ao consumidor final, vantagens estas já referidas anteriormente. Assim, é possível normalizar a estrutura funcional dos equipamentos, criar um ambiente de desenvolvimento independente do equipamento ou do fabricante, que vai por sua vez, simplificar o fabrico de equipamentos e reduzir os custos associados.

3.1.2 Program Organization Units

A norma apresentada anteriormente, traz a vantagem de conseguirmos realizar uma programação estruturada em elementos funcionais designados de POU's (*Program Organization Units*) e poder escolher a linguagem em que irá ser programada parte do projeto.

Os *Program Organization Units* (POU), são unidades básicas de uma aplicação de controlo. Estes estão divididos em três tipos diferentes [2]:

- *Function* (FUN);
- *Function Block* (FB);
- *Program* (PROG).

Segundo a norma, o POU é constituído por um cabeçalho e corpo estando as variáveis presentes no cabeçalho e as instruções para a execução no corpo, tal como está ilustrado na figura 3.1.

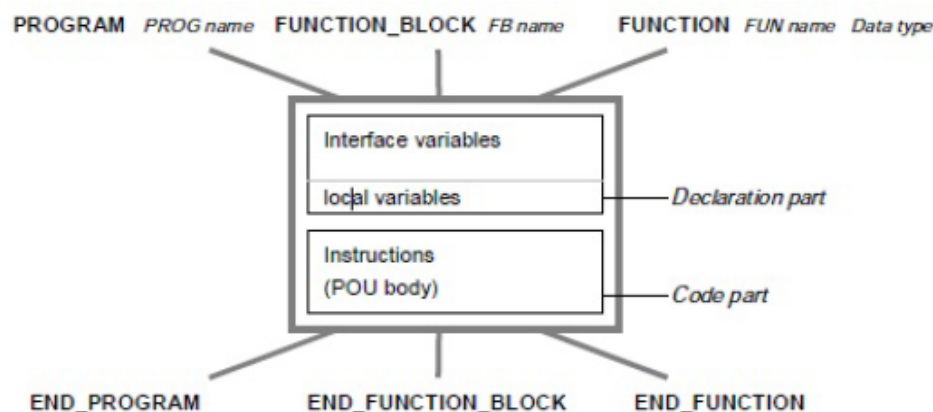


Figura 3.1: Estrutura comum dos 3 tipos de POU [2]

Os três diferentes tipos de POU referidos anteriormente, possuem uma hierarquia entre si. Os *Program* estão no topo da cadeia podendo incluir nas suas instruções *Function Blocks* e *Functions*, no entanto são os únicos que não podem incluir no seu corpo uma instância de outro PROG.

No nível seguinte da hierarquia vêm os *Function Blocks*, estes podem incluir no seu corpo FUN e outros FB ao contrário dos PROG que não permitem instâncias do mesmo tipo de POU. Os *Function Blocks* são os *Building Blocks* principais na estruturação de programas para os PLC's. Estes podem ser instanciados como variáveis, isto é, podemos ter variáveis em que o seu tipo é um FB. Depois de instanciado um *Function Block* pode ser usado como uma instância e chamado dentro do POU em que é declarado.

No último lugar da cadeia temos as *Function*, estas só podem chamar outras FUN, além disso as FUN diferem dos restantes tipos porque não possuem memória, isto é, o estado passado não é tido em conta na presente execução do programa.

Para além dos três tipos de POU já apresentados, a norma define alguns modelos de FUN e FB, os *Basic Building Blocks*. Estes são POU pré-definidos capazes de realizar funções de conversão, numéricas, entre outros.

3.1.3 Tipos de dados e Variáveis

3.1.3.1 Tipos de dados

O IEC 61131-3 também especifica vários tipos de dados, como por exemplo, o *Bool*, *Float*, *Byte*, *Integer*, entre outros que podem ser usados nos vários programas e blocos que são feitos.

Estas diferem, por exemplo, no número de bits ou no uso de sinais.

A tabela 3.1 mostra os vários tipos de dados definidos pela norma:

Tabela 3.1: Tipos de dados segundo a norma IEC 61131-3 [7]

Data Type	Size (bits)	Default Number
BOOL	1	False
BYTE	8	0
WORD	16	0
DWORD	32	0
LWORD	64	0
SINT/UINT	8	0
INT/UINT	16	0
DINT/UDINT	32	0
LINT/ULINT	64	0
REAL	32	0
LREAL	64	0
STRING	8(por caracter)	string vazia
WSTRING	16(por caracter)	string vazia
TIME	Implementação Dependente	T#0S
TIME_OF_DAY	Implementação Dependente	TOD#00:00:00
DATE	Implementação Dependente	D # 0001-01-01
DATE_AND_TIME	Implementação Dependente	DT # 0001-01-01-00:00:00

É também possível que o utilizador defina novos tipos de dados derivados dos tipos apresentados anteriormente. A criação de novos tipos de dados pode acontecer de diferentes formas:

- Direta - Cria um tipo de dados elementar com um particular valor inicial;
- Com intervalo - Cria um tipo de dados com um determinado limite. Por defeito, assume o valor mais baixo do intervalo;
- Enumeração - O tipo de dados pode assumir um valor de uma determinada lista;
- *Array* . Vários elementos do mesmo tipo de dados combinam e formam *array*;
- Estrutura - Vários elementos combinam e formam um tipo de dados.

Desta forma o utilizador pode definir qualquer tipo de dados que desejar. Os tipos de dados são fundamentais, visto que estão ligados a todas as variáveis do projeto. Isto é, todas as variáveis usadas num projeto estão definidas com um tipo de dados, esta característica oferece ao programa robustez, evitando assim a realização de operações inválidas como dividir um inteiro por uma *string*.

3.1.3.2 Variáveis

A norma também especifica os tipos de variáveis que podem ser internas, de entrada, saída ou entrada/saída.

As variáveis internas são usadas num POU mas o seu valor é irrelevante para o restante programa. As de entrada são alimentadas externamente por outras variáveis presentes noutro POU, acontecendo o contrário com as variáveis de saída, isto é, estas alimentam outras variáveis de outros POU.

A declaração da variável é efetuada tendo em conta os seguintes atributos [2]:

- Tipo de variável;
- Nome;
- Tipo de dados;
- Valor Inicial;
- Atributos.

Na tabela 3.2, mostra mais alguns tipos de variáveis, para além dos já mencionados anteriormente:

Tabela 3.2: Tipos de variáveis segundo a norma IEC 61131-3 [7]

Keyword	Description
VAR	Internal to POU variables
VAR_INPUT	Externally supplied, not modifiable within POU
VAR_OUTPUT	Supplied by POU to external entities
VAR_IN_OUT	Supplied by external entities, can be modified within POU
VAR_EXTERNAL	Supplied by configuration via VAR_GLOBAL, can be modified within POU
VAR_GLOBAL	Global variable declaration
VAR_ACCESS	Access path declaration

Quando são usadas variáveis remotas, como por exemplo, entradas e saídas de um PLC a sua definição é realizada recorrendo à expressão AT e concatenando os seguintes parâmetros:

"% + Localização + tamanho + um ou mais inteiros separados por '.' "

Existem ainda outros atributos para as variáveis, para além do AT, que estão apresentados na tabela 3.3:

Tabela 3.3: Atributos (qualificadores) das variáveis [7]

Attribute	Meaning
AT	Location Assignment
RETAIN	Constant variable (cannot be modified)
R_EDGE	Rising edge
F_EDGE	Falling Edge
READ_ONLY	Write-protected
READ_WRITE	Can be read and written to

A norma ainda refere que quando a representação inclui os inteiros separados por pontos finais, estes devem ser interpretados como um endereçamento numa lógica hierárquica, com o inteiro mais à esquerda a representar o nível mais alto e decrescendo da esquerda para a direita.

Na tabela 3.4, pode ser verificado quais os prefixos para a localização e tamanho das *directly represented variables*:

Tabela 3.4: Prefixos para a localização e tamanho das *directly represented variables* [7]

No	Prefix	Meaning
1	I	Input Location
2	Q	Output Location
3	M	Memory location
4	X	Single bit size
5	none	Single bit size
6	B	Byte (8 bits) size
7	W	Word (16 bits) size
8	D	Double word (32 bits) size
9	L	Long word (64 bits) size

3.1.4 Linguagens de Programação

Uma das principais vantagens da adoção do IEC 61131-3 é a uniformização/normalização das linguagens de programação para os PLC's.

A norma possui cinco linguagens:

- *Structured Text* (ST);
- *Instruction List* (IL);

- *Ladder Diagram* (LD);
- *Function Block Diagram* (FBD);
- *Sequential Function Chart* (SFC).

Destas cinco linguagens, duas delas são textuais, nomeadamente o ST (*Structured Text*) e o IL (*Instruction List*) e as restantes gráficas, que são o *Ladder* (LD), *Sequential Function Chart* (SFC) e o *Function Block Diagram* (FBD). Estas linguagens não possuem nenhuma restrição quanto ao uso de variáveis ou tipos de dados, além disso as linguagens conseguem interagir entre si sem haver qualquer problema.

O ST é uma linguagem de alto nível orientada a máquinas, semelhante ao Pascal e que consiste numa lista de procedimentos. Cada procedimento é processado determinando os valores que controlam o fluxo do programa. É uma linguagem que apresenta uma vasta gama de procedimentos abstratos e com grande complexidade.

Tem como vantagens, a possibilidade de uma forma de programação compacta, tendo as suas instruções claras e em blocos, e uma potente construção do fluxo do programa, permitindo a construção de estruturas complexas, como por exemplo, ciclos *While*, *For*, bem como a construção de expressões complexas, como operações matemáticas, processamento de dados, entre outras [2]. Além disto, apresenta algumas desvantagens como a sua tradução para código máquina não ser direta pois este é processado por um compilador.

A figura 3.2 mostra um excerto de código escrito em ST:

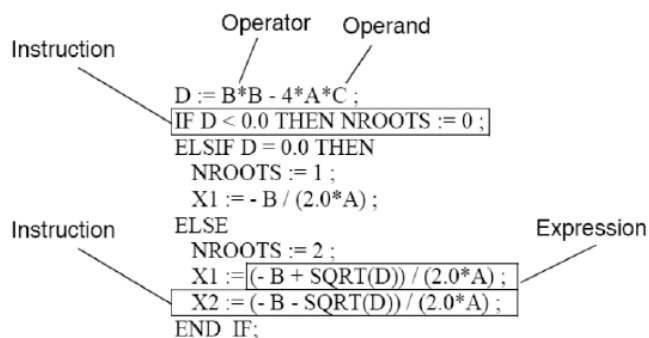


Figura 3.2: Excerto de código ST [3]

O IL é uma linguagem textual que ao contrário do ST é uma linguagem de baixo nível, semelhante à lista de instruções do *Assembly* usado na programação de microcontroladores. É muitas vezes usada na programação de sistemas de controlo de processos simples pois possui uma execução muito eficiente, isto é, devido à sua rapidez e baixo consumo de memória. Mas para além disto, esta tem algumas desvantagens, como a construção de estruturas e dificuldade em construir ciclos/sequências complexas, como o ciclo *For*. Se o programa for extenso, este também é difícil de ler e interpretar [2].

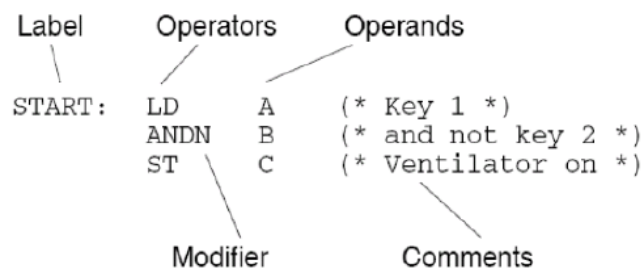


Figura 3.3: Excerto de código IL [3]

O LD é uma linguagem gráfica de baixo nível, análoga à construção de um circuito elétrico com relés. Esta representa o fluxo de energia da esquerda para a direita de um POU. É uma linguagem que era usada nos primeiros PLC equipados principalmente com relés, razão pela qual os seus símbolos são idênticos a circuitos elétricos. Estes têm a vantagem de possuir uma execução muito eficiente, em termos de rapidez e baixo consumo de memória, mas tal como o IL, é difícil construir estruturas complexas, bem como sequencias/ciclos complexos e se o programa for extenso existe dificuldade na sua análise [2].

Na figura 3.4 temos um excerto de código em LD:

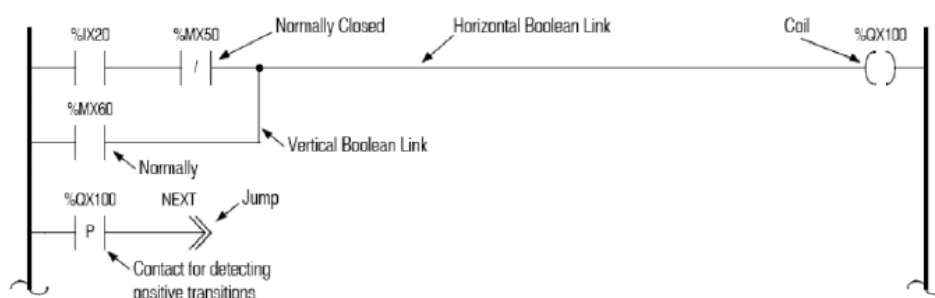


Figura 3.4: Excerto de código LD [3]

O FBD surgiu da área de processamento de sinal e descreve as interações entre os sinais de entrada e os sinais de saída que são processados através dos diferentes blocos. É uma linguagem gráfica de alto nível baseada no conceito de fluxo de sinal, incorporando uma programação orientada a objetos, e cada bloco implementa uma determinada ação de processamento, como pode ser visto na figura 3.5.

A sua utilização permite que seja uma linguagem modular.

Basicamente, o FBD é composto pela interligação de vários *Function Blocks*, *Function Blocks* estes que podem ser chamados por programas e por outros *Function Blocks* (instâncias).

As saídas de um bloco só podem ser conectadas às entradas de outro [2]. Além disso, não é possível conectar variáveis com diferentes tipos de dados. É uma linguagem poderosa e versátil que tem vindo a chamar a atenção do mundo da automação industrial.

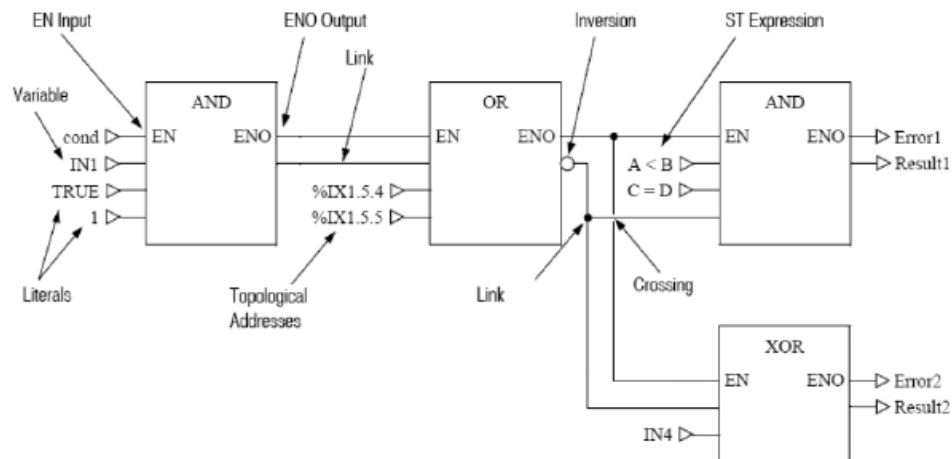


Figura 3.5: Excerto de código FBD [3]

O SFC é uma linguagem gráfica de alto nível, que descreve sequências de operações e interações entre processos paralelos, sequenciais e concorrentes. Esta fornece os meios para a estruturação do programa num conjunto de etapas separadas por transições, etapas estas que têm que ser programadas numa das outras linguagens disponíveis. Cada transição está associada a uma expressão que tem que ser satisfeita para que o programa continue com o seu fluxo. É uma linguagem com uma execução menos eficiente, pois é mais lenta e usa mais memória, mas como já foi dito, permite-nos uma boa organização do programa [2].

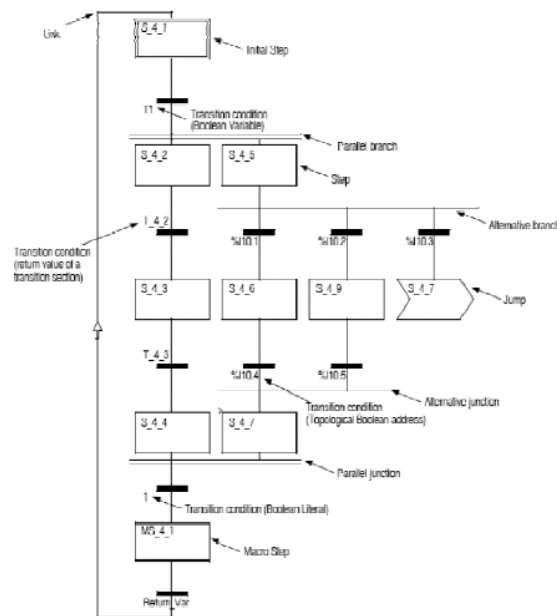


Figura 3.6: Excerto de código SFC [3]

Em conclusão, a norma IEC 61131-3, mostra uma nova forma de programar PLC's, onde entre várias vantagens, como a portabilidade do código para diferentes máquinas de diferentes fabrican-

tes, bem como a estruturação do código em funções e *Function Blocks*, que são componentes reutilizáveis, e quando bem entendidos e usados, permitem que as aplicações se tornem mais versáteis e modulares, dando assim uma redução do tempo de desenvolvimento, teste e implementação de sistemas de automação.

3.2 Beremiz

O Beremiz é um ambiente de desenvolvimento integrado, com a finalidade de desenvolver programas para PLC. Além de ser gratuito, multiplataforma e *open-source*, este segue a norma IEC 61131-3, mencionada anteriormente. A necessidade da criação deste ambiente teve como base diferentes fatores como [4]:

- Falta de soluções *open-source* nesta área;
- A dificuldade na portabilidade de programas desenvolvidos entre diferentes plataformas, apesar dos esforços de normalização;
- Os elevados custos de aquisição das licenças dos diversos fabricantes, dificultando assim o processo de aprendizagem;
- A dificuldade de acesso ao código fonte dos compiladores e do *runtime* impede a exploração dos mesmos e que se desenvolvam modos de operação pretendidos pelo utilizador.

O Beremiz foi desenvolvido de uma forma modular, tendo como componentes principais:

- *PLC Builder GUI* - Visão Geral dos projetos;
- *PLCOpen Editor* - Editor de programas;
- *MATIEC* - Compilador;
- *Plugins* - permitem a iteração com diversos tipos de tecnologias.

Nesta última versão do programa os componentes *PLC Builder GUI* e o *PLCOpen Editor* estão juntos numa só interface, logo serão apresentados em conjunto.

3.2.1 *PLC Builder GUI e PLCOpen Editor*

Na versão atual do Beremiz, o *PLC Builder GUI* deixou de estar separado do *PLCOpen Editor*, estando os dois componentes integrados numa só interface, o que possibilita uma mais fácil interação com os mesmos, ou seja, permite com que haja uma obtenção de informação mais rápida e mais intuitiva. Com a junção destas duas ferramentas conseguimos que várias interfaces sejam portáteis nas várias plataformas disponíveis.

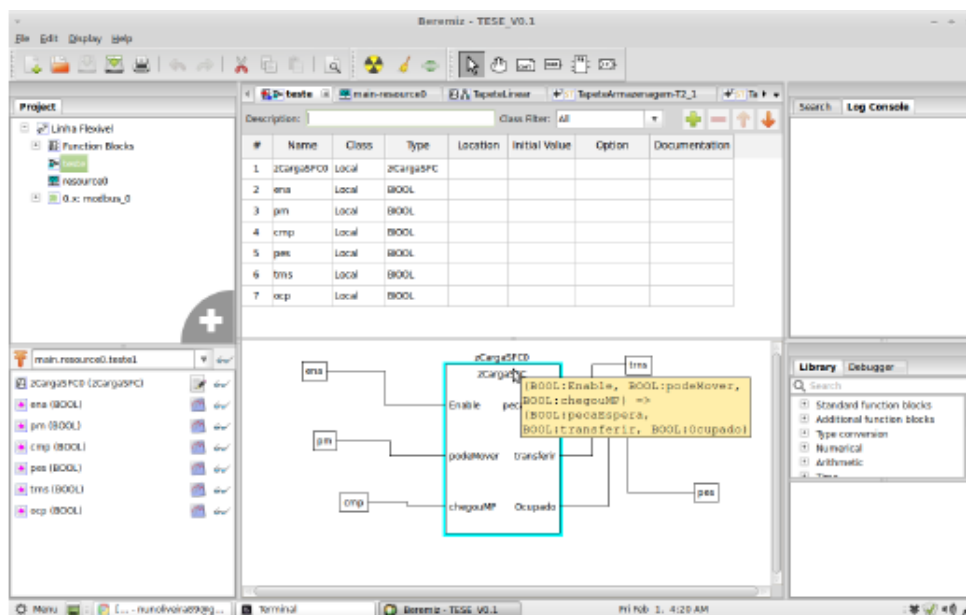


Figura 3.7: Interface Gráfica do Beremiz

A figura 3.7 mostra a interface gráfica do Beremiz. Todas as divisões podem ser ajustadas facilmente de acordo com as necessidades do utilizador. A divisão mais à esquerda está dividida em duas, na parte superior encontram-se árvores com o tipo de dados definidos pelo utilizador, os POU criados, as configurações, recursos e tarefas. Encontra-se um botão ("mais") que ao ser clicado mostra um menu onde se pode escolher o que se pretende acrescentar desde tipos de dados, POU que se pretende desenvolver até algum *plugin* que esteja disponível.

Na parte inferior estão presentes as instâncias presentes no projeto. Esta divisão apresenta uma visão global de todo o projeto que vai desde os tipos de dados até *plugins* existentes. Na divisão central é possível a escrita das instruções que cada POU deve executar.

Como o Beremiz segue a norma IEC 61131-3, a escrita das instruções a ser executadas no POU podem ser feitas usando as linguagens de programação disponíveis pela norma, podendo ser escritas de forma textual utilizando por exemplo o ST ou recorrendo a uma linguagem gráfica como o FBD ou SFC. Na parte superior desta divisão, é possível adicionar ou remover as variáveis associadas ao POU ativo.

Na divisão à direita está presente uma biblioteca de funções padrão ou já definidas pelo utilizador que podem ser usadas a qualquer momento, para isso basta que a função pretendida seja arrastada para a zona de escrita. Por fim na divisão superior encontra-se uma *Log Console* que contém diversas informações quanto à compilação e execução do programa. Os projetos são guardados em formato *XML*, seguindo a estrutura definida no *XML Official Schema* do comité técnico TC6 da organização PLCOpen.

3.2.2 Compilador Matiec

O compilador MATIEC tem como principal ação consumir o resultado da conversão textual de um determinado projeto IEC 61131-3. Basicamente o que o compilador faz é pegar no projeto em IEC 61131-3 e converte-lo para código na linguagem C. Dito isto, o código produzido em C é organizado da forma representada na figura 3.8.

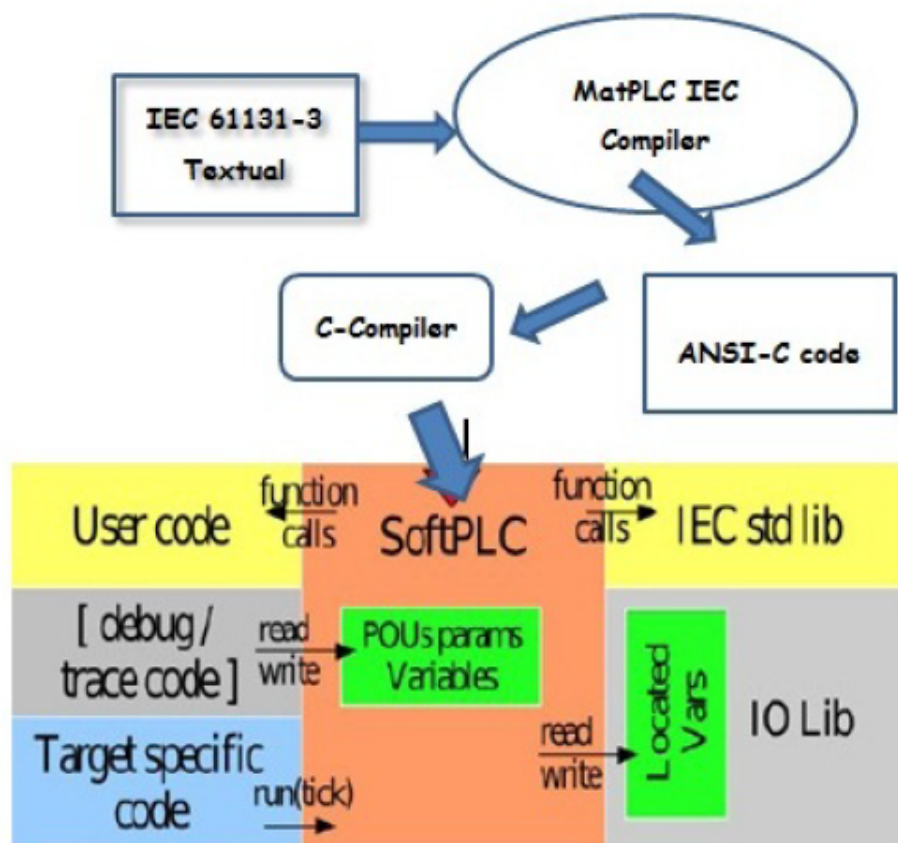


Figura 3.8: Etapas gerais de compilação e organização do código [4]

O compilador funciona em quatro etapas:

- Analisador Léxico
- Analisador de sintaxe
- Analisador da semântica
- Gerador de código

Através da figura é possível ver-se que tanto as variáveis como os parâmetros dos POU são declarados como estruturas C em árvore, enquanto as variáveis onde se define uma localização são declaradas como externas.

O algoritmo de controlo do SoftPLC é executado por iniciativa de um modulo próprio (*Target Specific Code*), responsável por gerir o relógio específico da plataforma alvo, e gerar interrupções cadenciadas para a execução das tarefas.

O programa acede a um conjunto de funções e *Function Blocks*, sejam eles da biblioteca padrão (*IEC std lib*) ou criados pelo utilizador (*User Code*), que se encontram definidos num módulo próprio e que recebem como parâmetros as estruturas C.

Dispõem ainda de outro modulo, que consiste em consumir o estado atual de todos os parâmetros do programa, reproduzindo esse estado de funcionamento ao utilizador em *runtime*, através do *PLCOpen Editor*.

É importante salientar que o Beremiz apresenta uma restrição face ao disposto na norma IEC 61131-3. Esta refere no seu modelo de programação que uma dada configuração contém um ou mais recursos, cada um dos quais contém um ou mais programas executados sob o controlo de zero ou mais tarefas. No Beremiz, é possível instanciar um único recurso onde podemos ter vários programas sob controlo de várias tarefas, sendo que esses programas podem, e aqui sim em conformidade com a norma, instanciar as funções ou FB que se pretenda.

3.2.3 Plugins

Os *plugins* permitem de uma forma generalizada, que exista comunicação entre o *SoftPLC* e o mundo exterior. Assim sendo, existem ações de controlo que são necessárias para a interligação com diversos dispositivos (físicos e lógicos), que podem ter um contacto direto com o processo que se pretende controlar. Estes poderão ser de vários tipos como: sensores, atuadores, interfaces, entre outros.

Além destes *plugins* serem compostos por uma interface com o utilizador como também por uma componente de código em C, é importante mencionar que o seu uso está definido através de uma simbologia própria para a associação das variáveis aos tais dispositivos na estrutura de entrada, saída ou até memória do PLC. Isto consoante o referido na norma IEC 61131-3 sobre as *directly represented variables*, que podem ser vistas na tabela 3.4.

3.3 RS232

3.3.1 Visão Geral

O RS, no protocolo vem da abreviação de *Recommended Standard*, esta relata uma normalização de uma interface comum para a comunicação de dados entre equipamentos, criada no início do anos 60, por um comité conhecido atualmente como *Electronic Industries Association* (EIA). Naquele tempo, a comunicação de dados compreendia a troca de dados digitais entre um computador central (*mainframe*) e terminais de computadores remotos, ou entre dois terminais sem o envolvimento do computador. Estes dispositivos eram conectados através da linha telefónica e consequentemente necessitavam de um modem em cada lado para ser realizada a descodificação dos sinais.

Dessas ideias nasceu o protocolo RS232, que é um protocolo de comunicação série. Este especifica as tensões, temporizações e funções dos sinais, conexões mecânicas, é um protocolo para troca de informação.

Este protocolo já sofreu várias atualizações sendo que a mais recente foi introduzida em 1991.

Embora tenha sofrido poucas alterações, muitos fabricantes adotaram diversas soluções mais simplificadas que tornaram impossível a simplificação da normalização proposta. As maiores dificuldades encontradas pelos utilizadores do protocolo RS232 incluem pelo menos um dos fatores que se seguem [10]:

- Ausência ou conexão errada de sinais de controlo que resultam no *overflow* ou falhas na comunicação;
- Função incorreta de comunicação, que pode resultar na inversão das linhas de transmissão e recepção, bem como a inversão de uma ou mais linhas de controlo, isto é, *handshaking*

Em conclusão, a interface RS232 tem as seguintes características [6]:

- Comunicação ponto-a-ponto;
- Transmissão em tensão não-balanceada;
- Transmissão série: assíncrona e síncrona;
- Funcionamento em modos: simplex, half-duplex ou duplex/full-duplex;
- Velocidades baixas/médias (max. 115 Kb/s, PC);
- Distâncias de transmissão curtas (max. 20 m);
- Sinais de dados: Transmissão/Recepção e um canal de transmissão secundário;
- Sinais de controlo: Controlo de fluxo (*handshake*), interface com o modem, Teste e verificação de operacionalidade dos equipamentos
- Fortemente suportado, universal e standardizado.

A transmissão de mensagens neste protocolo, é realizada através da divisão das mesmas em mensagens menores que são transmitidas sequencialmente. A transmissão série converte a mensagem e envia um bit de cada vez pelo canal de comunicação, sendo que cada bit representa uma parte da mensagem. Quando os bits chegam ao destino, estes são reorganizados de forma a comporem a mensagem original.

3.3.2 Modos de Transmissão

Como referido anteriormente o RS232 pode funcionar em 3 modos diferentes, o modo *simplex*, *half-duplex* e *duplex/full-duplex*.

Basicamente, estes modos de funcionamento definem como é que a informação vai navegar no canal de comunicação. Um canal de comunicação é o caminho sobre o qual a informação flui. Este pode ser definido por uma linha física, por exemplo, um fio que liga os dispositivos de comunicação, um rádio, laser ou outra fonte de energia radiante.

Falando agora das diferenças dos modos de funcionamento:

- **Simplex** - modo em que a informação só circula num sentido, tomemos como exemplo, uma estação de rádio pois ela transmite sempre o sinal para os ouvintes mas nunca é permitido o contrário;
- **Half-Duplex** - já é um modo que permite que o sentido da transmissão possa ser realizado nos dois sentidos. As mensagens podem fluir nas duas direções mas nunca ao mesmo tempo, um bom exemplo de um canal *half-duplex* é uma chamada telefónica, em que só conseguimos ter uma conversa se falar uma pessoa de cada vez;
- **Duplex/Full-Duplex** - é um modo em que é permitido a troca simultânea de mensagens. Este pode ser visto como 2 canais *simplex*, um canal direto e um outro na direção oposta ligados nos mesmos pontos

3.3.3 Especificação Funcional e Mecânica

Se a norma completa for implementada, o equipamento que faz o processamento dos sinais é chamado DTE (*Data Terminal Equipment*) que é usualmente um computador ou um terminal que tem um conector DB25 macho e utiliza 22 dos 25 pinos disponíveis para sinais ou terra. O equipamento que faz a conexão, normalmente uma interface com a linha telefónica, é chamado de DCE (*Data Circuit-terminating Equipment*), que é usualmente um modem, este tem um conector DB25 fêmea e usa os mesmos 22 pinos que o DTE utiliza. Um cabo de conexão entre dispositivos DTE e DCE contém ligações em paralelo, o que faz com que não seja necessário mudanças na conexão dos pinos.

No anexo A são apresentadas as ligações entre DTE e DCE, bem como entre dois DTE. Está também apresentado a função dos pinos mais usados na comunicação série.

3.4 Function Blocks for Motion Control

Neste capítulo serão apresentados os *Function Blocks* desenvolvidos pela PLCOpen, explicando algumas características dos mesmos e apresentando os FB que foram escolhidos para serem desenvolvidos.

3.4.1 Visão Geral

O mercado de controlo de movimento (*Motion Control*) disponibiliza uma grande variedade de sistemas e soluções incompatíveis. Em empresas onde sistemas diferentes são usados, esta incompatibilidade induz custos elevados para o utilizador final, a aprendizagem é difícil, a engenharia torna-se difícil e o crescimento do mercado abranda.

A normalização certamente faria com que esses fatores negativos diminuíssem. A normalização mencionada não significa apenas as linguagens de programação apresentadas na norma IEC 61131-3, mas também a normalização da interface para as diferentes soluções de controlo de movimento. Desta forma, a programação dessas soluções para o controlo de movimento serão menos dependentes do *hardware*. A reutilização da aplicação de *software* é aumentada e os custos envolvidos no treino e suporte serão reduzidos.

Para a resolução deste problema a PLCOpen criou a *Motion Control Task Force*, que definiu a interface normalizando assim os *Function Blocks for Motion* [5].

Estes *Function Blocks* são aplicáveis nas linguagens do IEC 61131-3 considerando os seguintes fatores [5]:

- Simplicidade - facilidade de utilização, para o programador da aplicação e para a instalação e manutenção;
- Eficiência - no número de *Function Blocks*, direcionada para a eficiência no *Design* e entendimento;
- Consistência - conformidade com a norma IEC 61131-3;
- Universalidade - independente do *hardware*;
- Flexibilidade - para extensões futuras ou aumento da amplitude da aplicação;
- Plenitude - não é obrigatória mas é suficiente.

A definição deste conjunto de FB preocupa-se com a granularidade e modularidade dos *Function Blocks* standardizados. Os extremos são um FB por eixo em relação a uma funcionalidade a nível de comando. Os objetivos acima mencionados podem ser conseguidos facilmente através de um *design* modular dos FB, pois a modularidade cria um nível elevado de escalabilidade, flexibilidade e reconfiguração.

Na figura 3.9, são apresentados os vários *Function Blocks* definidos e divididos em administrativos e de movimento, isto é, FB que não acionam movimento e os que o induzem:

<i>Administrative</i>		<i>Motion</i>	
<i>Single Axis</i>	<i>Multiple Axis</i>	<i>Single Axis</i>	<i>Multiple Axis</i>
MC_Power	MC_CamTableSelect	MC_Home	MC_CamIn
MC_ReadStatus		MC_Stop	MC_CamOut
MC_ReadAxisError		MC_Halt	MC_GearIn
MC_ReadParameter		MC_MoveAbsolute	MC_GearOut
MC_ReadBoolParameter		MC_MoveRelative	MC_GearInPos
MC_WriteParameter		MC_MoveAdditive	MC_PhasingAbsolute
MC_WriteBoolParameter		MC_MoveSuperimposed	MC_PhasingRelative
MC_ReadDigitalInput		MC_MoveVelocity	MC_CombineAxis
MC_ReadDigitalOutput		MC_MoveContinuousAbsolute	
MC_WriteDigitalOutput		MC_MoveContinuousRelative	
MC_ReadActualPosition		MC_TorqueControl	
MC_ReadActualVelocity		MC_PositionProfile	
MC_ReadActualTorque		MC_VelocityProfile	
MC_ReadAxisInfo		MC_AccelerationProfile	
MC_ReadMotionState			
MC_SetPosition			
MC_SetOverride			
MC_TouchProbe			
MC_DigitalCamSwitch			
MC_Reset			
MC_AbortTrigger			
MC_HaltSuperimposed			

Figura 3.9: Lista dos vários *Function Blocks* disponíveis [5]

3.4.2 Modelo

Os *Function Blocks* que foram desenvolvidos pela PLCOpen têm o objetivo de controlar eixos via as linguagens de programação que estão definidas na norma IEC 61131-3. Foi decidido pela *Motion Control Task Force* que não seria prático que todos os aspectos de um eixo fossem encapsulados em apenas um *Function Block*. A solução escolhida foi disponibilizar um conjunto de *Function Blocks* comandados e que possuem uma referência para o eixo, por exemplo o tipo de dados "Axis", que oferece flexibilidade, facilidade de utilização e reutilização.

As implementações baseadas na norma IEC 61131-3, por exemplo através de *Function Blocks* e SFC, serão focadas para a interface (*look-and-feel/proxy*) dos FB.

De seguida será apresentado o diagrama de estados que define o comportamento do eixo quando múltiplos *Motion Control Function Blocks* estão "simultaneamente" ativos.

3.4.2.1 Diagrama de Estados

Como mencionado anteriormente, este diagrama de estados representa o comportamento do eixo quando temos múltiplos MC_FB em funcionamento. Esta combinação de perfis de movimento são úteis para a construção de um perfil mais complicado e na forma como lidar as exceções que podem aparecer dentro de um programa.

Basicamente, o que a regra impõe é que os comandos que geram o movimento sejam realizados sequencialmente, mesmo que o PLC tenha a capacidade de processamento paralelo.

O eixo está sempre num dos estados definidos, qualquer comando que induza movimento e que cause uma transição irá fazer com que haja uma mudança no estado do eixo. A mudança de estado é refletida "imediatamente" quando o comando de movimento é realizado.

O Diagrama de Estados é considerado como uma camada de abstração do estado onde está na realidade o eixo, é comparável com a imagem dos I/O num ciclo de um programa de um PLC.

De seguida na figura 3.10 é apresentado o Diagrama de Estados usado pelos *Motion Control Function Blocks* [5].

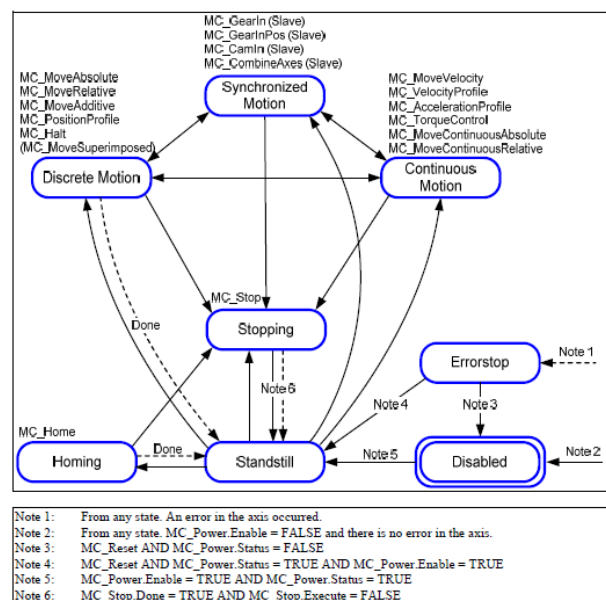


Figura 3.10: Diagrama de Estados dos *Motion Control Function Blocks* [5]

O diagrama apresentado está focado em apenas um eixo. Para FB de múltiplos eixos, estes podem ser observados de um ponto de vista do diagrama, como múltiplos "eixos únicos" todos em estados específicos. Por exemplo, o CAM-Master pode estar no estado "*ContinuousMotion*" e o escravo correspondente no estado "*SynchronizedMotion*".

Explicando melhor o diagrama, as setas a cheio representam as possíveis transições de estado devido a um comando, enquanto que as setas a tracejado são usadas para representar transições de estado que ocorrerem quando um comando terminou ou algo de errado aconteceu. Os comandos de movimento que fazem com que o eixo troque de estado estão mencionados em cima do estado correspondente.

Falando agora dos principais estados do diagrama:

- *Disable* - descreve o estado inicial do eixo. Neste estado o movimento do eixo não é influenciado pelo FB's. Para sair deste estado o FB *MC_Power* tem de ser chamado com o *Enable* = TRUE, o estado passa do *Disable* para o estado *StandStill*;
- *ErrorStop* - é um estado com uma prioridade elevada e aplicável quando ocorre um erro, podendo a transição para este estado ser feita de qualquer um dos outros estados. A intenção deste estado é que o eixo pare, se possível. Para sair deste estado é necessário que seja feito um *reset*, pois enquanto este não for feito nenhum dos outros comandos será aceite. A

transição para *ErrorStop* apenas se refere a erros no eixo ou no controlo do eixo não de erros nas instâncias dos *Function Blocks*;

- *StandStill* - é o estado em que o eixo está operacional para o movimento e se encontra parado à espera que seja acionado um comando para que o eixo se movimente.

3.4.2.2 Tratamento de erros

Todo o acesso para o controlo de movimento é feito através dos *Function Blocks*, internamente estes fornecem uma verificação básica de erros na entrada.

Por exemplo, se o *MaxVelocity* possuir o valor 6000, e a entrada *Velocity* do *Function Block* está a 10000, só existem duas soluções ou o sistema abrande ou este vai gerar um erro [5].

Para o tratamento dos erros, tanto o método centralizado ou o descentralizado podem ser usados, pois ambos são possíveis quando os *Motion Control Function Blocks* são usados.

Quanto ao método centralizado este é usado para simplificar a programação do *Function Block*. A reação ao erro é independente da instância na qual o erro ocorreu [5].

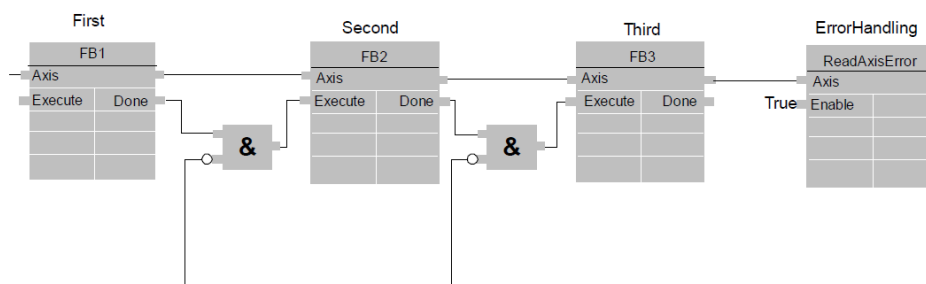


Figura 3.11: Exemplo de FB com o tratamento de erros centralizado [5]

Quanto ao tratamento de erros descentralizado, é um pouco diferente, este dá a possibilidade de ter diferentes reações ao erro ocorrido, dependendo do *Function Block* onde o erro ocorreu.

A figura 3.12, mostra um exemplo de FB onde é o usado o tratamento descentralizado:

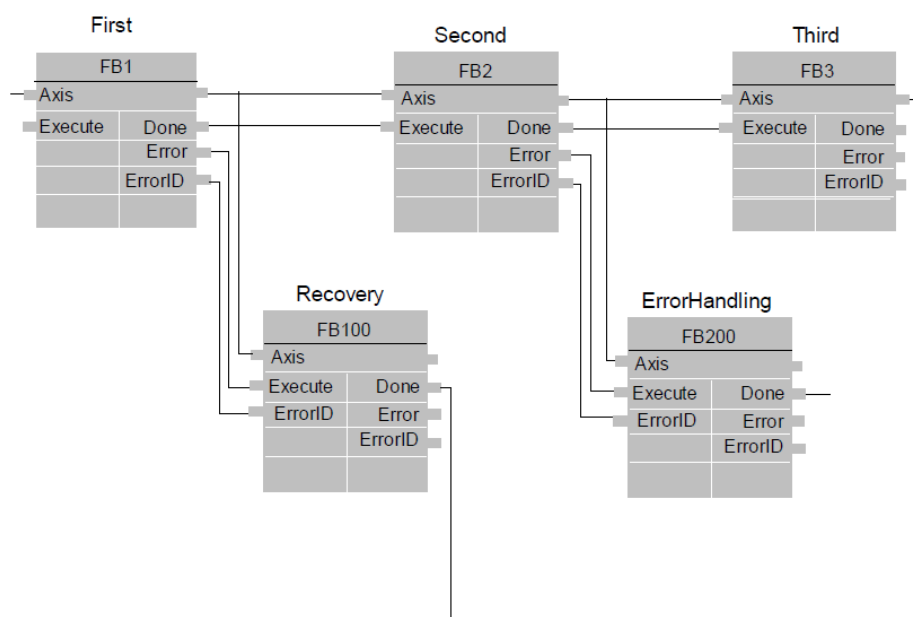


Figura 3.12: Exemplo de FB com o tratamento de erros descentralizado [5]

3.4.2.3 Buffered Modes

Alguns destes *Function Blocks* possuem uma entrada chamada "*BufferMode*". Esta entrada permite que os FB trabalhem ou num "*Non-buffered mode*", que é o comportamento por defeito, ou num "*Buffered mode*". A diferença entre estes dois modos verifica-se quando o FB deve iniciar a sua ação, isto é, quando um comando está no modo "*non-buffered*" este é realizado imediatamente, mesmo que este interrompa outro movimento, após isto o *buffer* é limpo. Se um comando está no modo "*Buffered*", este espera até que o FB que está ativo naquele momento coloque a sua saída a um.

Neste último modo, temos várias opções que são as seguintes [5]:

- *Aborting* - Modo que está escolhido por defeito, consiste em que FB seguinte aborte qualquer movimento que está a ser executado e o comando afeta o eixo/motor imediatamente e limpa o *buffer*;
- *Buffered* - Consiste em que o FB seguinte afete o eixo assim que o anterior coloque a um a sua saída *Done*;
- *BlendingLow* - O FB seguinte controla o eixo depois do anterior ter terminado, mas o eixo não parará entre os movimentos. A velocidade com que este rodará será a menor entre os dois comandos.
- *BlendingPrevious* - Funciona como o anterior, só que a velocidade escolhida será a velocidade do FB anterior e não a do FB que vai realizar o comando.

- *BlendingNext* - Ao contrário do anterior este faz com que a velocidade do eixo seja a velocidade do FB que vai realizar o comando;
- *BlendingHigh* - Consiste em realizar o comando proposto à maior velocidade entre a do FB que acabou de realizar um comando e a do FB que vai realizar o comando seguinte.

3.4.2.4 Apresentação dos *Function Blocks* escolhidos

Como já foi mencionado, foram escolhidos alguns dos *Motion Control Function Blocks* definidos pela PLCOpen para serem desenvolvidos e interligados de forma a atingir o objetivo proposto. De seguida são apresentados os FB escolhidos [5]:

- **MC_Power** - Este FB tem como função controlar se o dispositivo está ON/OFF, isto é, se está ligado e desligado. Na figura 3.13 vê-se a constituição do FB.

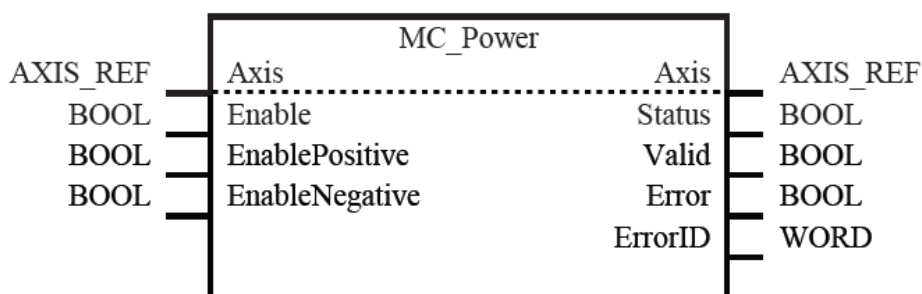


Figura 3.13: MC_Power FB [5]

Especificando as entradas do FB:

- *AXIS_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Enable* - se estiver a *TRUE*, o sistema está ligado;
- *EnablePositive* - Entrada opcional em que se o *Enable* for *TRUE* permite o movimento na direção positiva;
- *EnableNegative* - Entrada opcional em que se o *Enable* for *TRUE* permite o movimento na direção negativa;

Em relação às saídas:

- *AXIS_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Status* - Como o nome indica, esta saída dá o estado efetivo do sistema;
- *Valid* - Se for *TRUE*, podemos ter conjuntos de saídas disponíveis para o FB;

- *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_Home** - Este FB comanda o eixo a "procurar uma casa", isto é, o eixo vai para uma espécie de posição de referência. A figura 3.14, mostra quais as entradas e saídas deste *Function Block*.

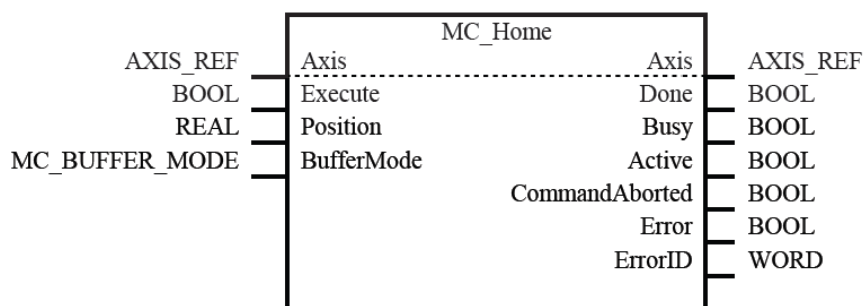


Figura 3.14: MC_Home FB [5]

Entradas do FB:

- *AXIS_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *Position* - É a posição quando o sinal de referência é detetado;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Done* - Como o nome indica, esta saída diz-nos que o FB concluiu a sua função;
- *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
- *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
- *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
- *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
- *ErrorID* - Saída opcional, identificação do erro.

- **MC_Stop** - Este *Function Block* tem como função parar qualquer ação de movimento que esteja a ocorrer e transferir o eixo para o estado "*Stopping*" do diagrama de estados, ver sub-capítulo 3.4.2.1. Este FB é mais usado para paragens de emergência ou situações especiais. Enquanto a entrada *Execute* se mantiver a *TRUE* ou se o motor ainda não tiver atingido a velocidade zero, este mantém-se no estado "*Stopping*" e só transita quando a entrada *Execute* for *False* e a saída *Done* for *TRUE*, transitando assim para o estado "*StandStill*".

A figura 3.15, mostra quais as entradas e saídas do FB.

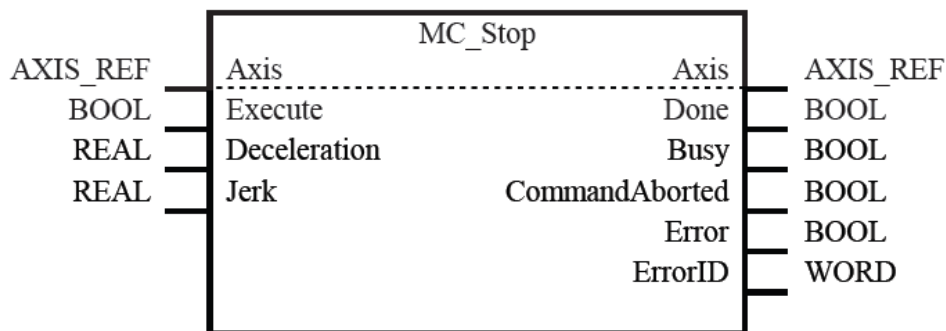


Figura 3.15: MC_Stop FB [5]

Entradas do FB:

- *Axis_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.

Saídas do FB:

- *Axis_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Done* - Como o nome indica, esta saída diz-nos que o FB concluiu a sua função;
- *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
- *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
- *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
- *ErrorID* - Saída opcional, identificação do erro.

- **MC_Halt** - Este *Function Block* é usado para comandar uma paragem de movimento controlada, ou seja, é usado para parar o eixo em condições normais de funcionamento. Em termos do diagrama de estados, o MC_Halt faz com que o eixo se mude para o estado "*DiscreteMotion*", até que a velocidade seja zero. Quando a saída *Done* for a *TRUE*, o eixo passa para o estado "*StandStill*".

A figura 3.16 mostra a constituição do MC_Halt.

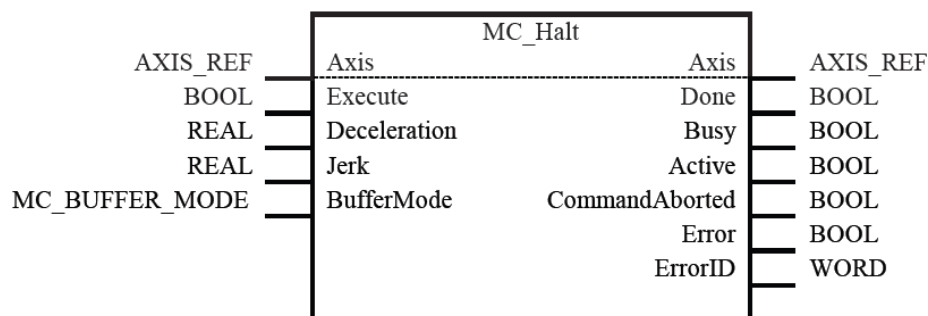


Figura 3.16: MC_Halt FB [5]

Entradas do FB:

- *Axis_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *Axis_REF* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Done* - Como o nome indica, esta saída diz-nos que o FB concluiu a sua função;
- *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
- *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
- *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
- *Error* - Vai a 1, se algum erro acontecer no *Function Block*;

- *ErrorID* - Saída opcional, identificação do erro.
- **MC_MoveAbsolute** - Este FB comanda o eixo a realizar um movimento para uma posição especificada.

A figura 3.17, mostra quais as suas entradas e saídas.

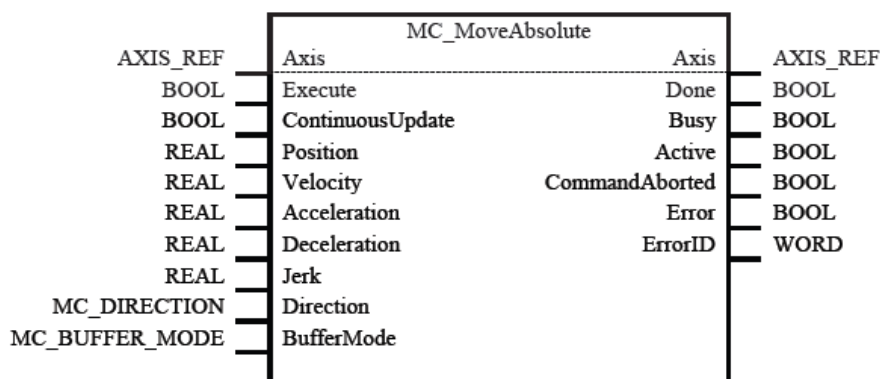


Figura 3.17: MC_MoveAbsolute FB [5]

Entradas do FB:

- *AXIS_REF*
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *Position* - Posição para onde o FB vai;
- *Velocity* - Valor máximo que a velocidade pode atingir;
- *Acceleration* - Entrada opcional, indica o valor da aceleração, é sempre positivo;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.
- *Direction* - Pode ter 1 de 4 valores: *mcPositiveDirection*, *mcShortestWay*, *mcNegativeDirection*, *mcCurrentDirection*;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF*

- *Done* - Como o nome indica, esta saída diz-nos que o FB concluiu a sua função, neste caso é *TRUE* quando chega à posição escolhida;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_MoveRelative** - Este *Function Block* realiza um movimento controlado de uma distância especificada relativamente a uma dada posição, ou seja, a partir de uma posição, este faz com que o eixo se movimente uma certa distância.

A figura 3.18, mostra quais as suas entradas e saídas.

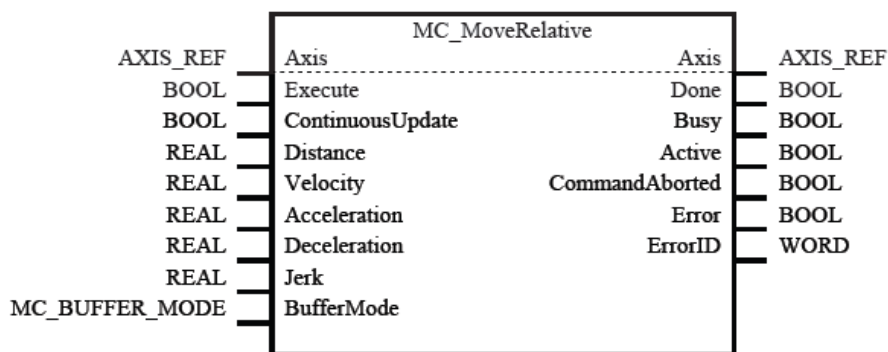


Figura 3.18: MC_Relative FB [5]

Entradas do FB:

- *AXIS_REF*
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *Distance* - Distancia que queremos percorrer;
- *Velocity* - Valor máximo que a velocidade pode atingir;
- *Acceleration* - Entrada opcional, indica o valor da aceleração, é sempre positivo;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;

- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF*
 - *Done* - Como o nome indica, esta saída diz-nos que o FB concluiu a sua função, neste caso é *TRUE* quando chega à posição escolhida;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_MoveSuperimposed** - Este *Function Block* comanda um movimento controlado de uma distância relativa adicional a um movimento existente.

O movimento existente não é interrompido, mas sobreposto pelo movimento adicional.

Na figura 3.19 vemos a constituição do FB.

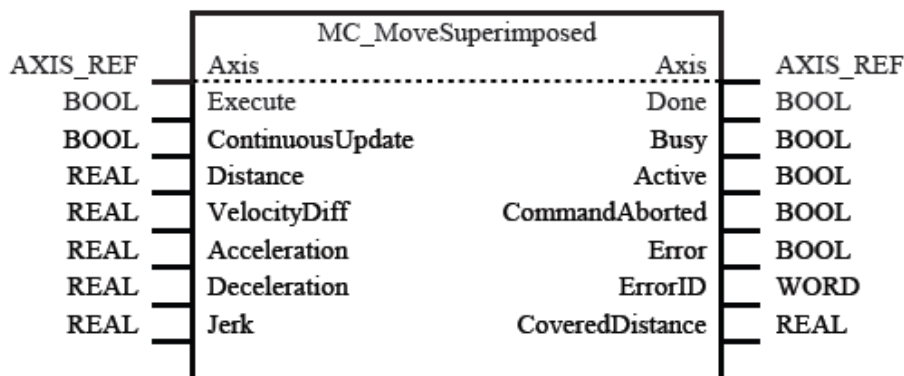


Figura 3.19: MC_MoveSuperimposed FB [5]

Entradas do FB:

- *AXIS_REF*
- *Execute* - Quando houver um *rising edge* inicia o movimento;

- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *Distance* - Distancia que queremos percorrer;
- *VelocityDiff* - Diferença de velocidade entre o movimento adicional e o movimento em execução;
- *Acceleration* - Entrada opcional, indica o valor da aceleração, é sempre positivo;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.

Saídas do FB:

- *AXIS_REF*
 - *Done* - Como o nome indica, esta saída diz-nos que o FB concluiu a sua função, neste caso é *TRUE* quando chega à posição escolhida;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
 - *CoveredDistance* - Saída que mostra continuamente a distância percorrida pelo FB desde que foi acionado.
- **MC_HaltSuperimposed** - Este FB funciona como o MC_Halt apresentado anteriormente, mas com uma exceção, em vez de comandar uma paragem a todos os movimentos apenas comanda a paragem de todos os movimentos *superimposed* presentes no eixo.

A imagem [3.20](#) mostra a constituição do FB.

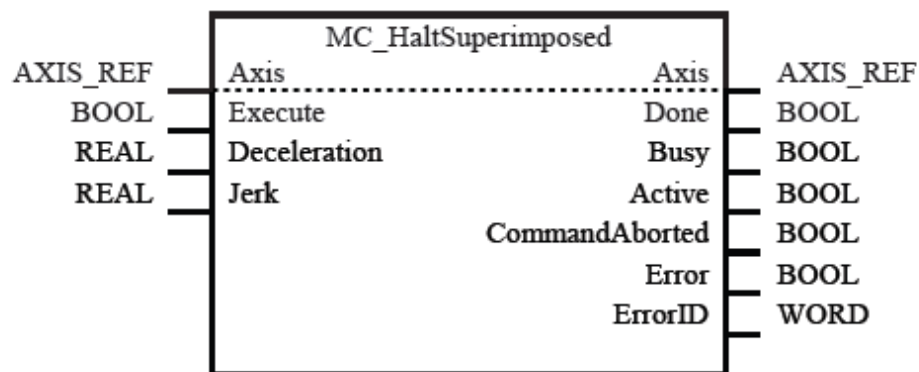


Figura 3.20: MC_HaltSuperimposed FB [5]

Entradas do FB:

- *Axis* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *Axis* - esta entrada/saída dá a referência para o eixo, isto é, indica qual o eixo a ser controlado;
 - *Done* - Vai a *TRUE*, quando os movimentos *Superimposed* são interrompidos;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_MoveVelocity** - Este *Function Block* realiza um movimento "infinito", isto é, contínuo a uma velocidade específica definida pelo utilizador.

A figura 3.21 mostra as entradas e saídas do FB.

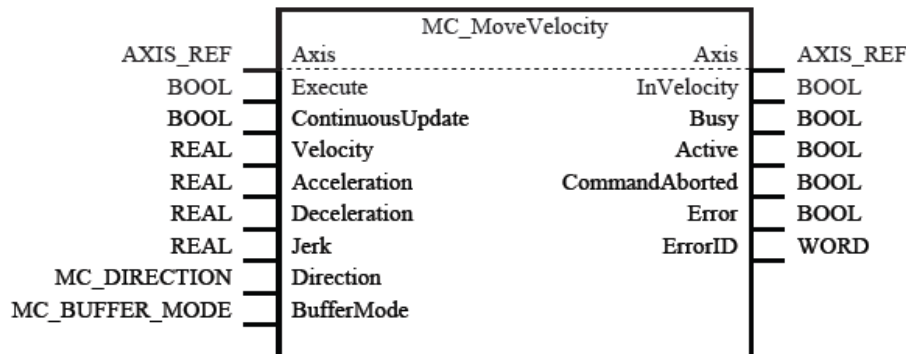


Figura 3.21: MC_HaltSuperimposed FB [5]

Entradas do FB:

- **AXIS_REF**
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *Velocity* - Valor máximo que a velocidade pode atingir;
- *Acceleration* - Entrada opcional, indica o valor da aceleração, é sempre positivo;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.
- *Direction* - Pode ter 1 de 3 valores: *mcPositiveDirection*, *mcNegativeDirection*, *mcCurrentDirection*;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- **AXIS_REF**
- *InVelocity* - Vai a *TRUE*, quando a velocidade atinge a velocidade especificada;
- *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
- *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;

- *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_MoveContinuousAbsolute** - Este *Function Block* comanda um movimento controlado para uma posição específica acabando o movimento na velocidade escolhida. Tome-mos como exemplo, um laser que é usado para cortar uma peça, o que este FB vai fazer é movimentar o laser à velocidade máxima até chegar à posição inicial de corte, verifica se tem a velocidade ao qual vai realizar o corte quando está na posição inicial. Liga o laser e vai pela peça a velocidade constante (*EndVelocity*) até à posição final de corte, chegando a este local desliga o laser e volta para a posição de espera à velocidade máxima.

A figura 3.22 mostra a constituição do FB.

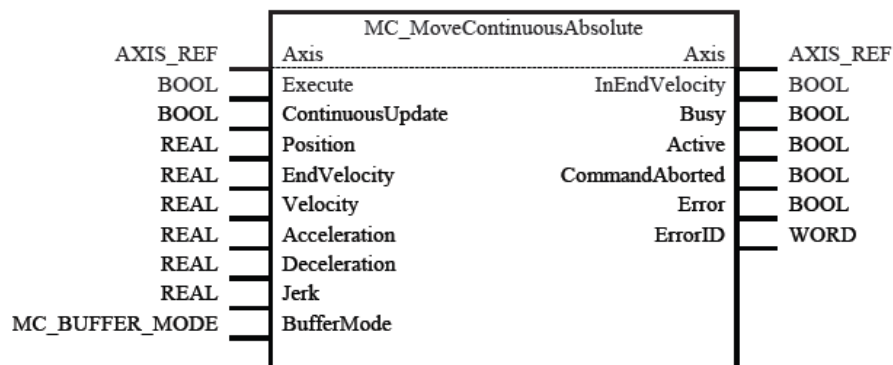


Figura 3.22: MC_MoveContinuousAbsolute FB [5]

Entradas do FB:

- *AXIS_REF*
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *Position* - Posição para onde o FB vai;
- *EndVelocity* - Valor da velocidade final, tomando como exemplo a velocidade de corte;
- *Velocity* - Valor máximo que a velocidade pode atingir;
- *Acceleration* - Entrada opcional, indica o valor da aceleração, é sempre positivo;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.

- *Direction* - Pode ter 1 de 4 valores: *mcPositiveDirection*, *mcShortestWay*, *mcNegativeDirection*, *mcCurrentDirection*;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF*
 - *InEndVelocity* - Vai a *TRUE*, quando atinge a posição desejada e está a movimentar-se com a *EndVelocity*;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_MoveContinuousRelative** - Este FB comanda um movimento relativo a uma distância especificada, acabando o movimento com a velocidade escolhida. Basicamente funciona como o FB anterior, só que em vez de termos uma posição absoluta temos uma distância que este vai ter que percorrer a uma velocidade constante.

Na figura 3.23 podemos ver as entradas e saídas constituintes deste FB.

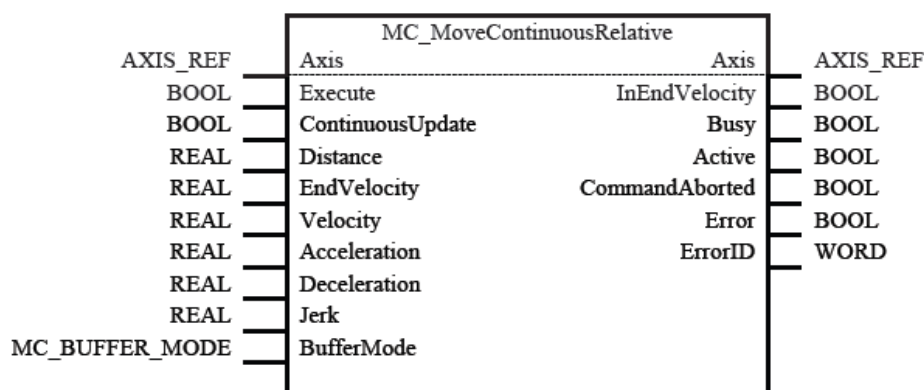


Figura 3.23: MC_MoveContinuousRelative FB [5]

Entradas do FB:

- *AXIS_REF*

- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *Distance* - Distância que se quer percorrer;
- *EndVelocity* - Valor da velocidade final, tomando como exemplo a velocidade de corte;
- *Velocity* - Valor máximo que a velocidade pode atingir;
- *Acceleration* - Entrada opcional, indica o valor da aceleração, é sempre positivo;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF*
 - *InEndVelocity* - Vai a *TRUE*, quando atinge a posição desejada e está a movimentar-se com a *EndVelocity*;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_TorqueControl** - Este FB exerce continuamente um binário ou a força da magnitude especificada. Esta magnitude é aproximada através de uma rampa definida (“*Torque-Ramp*”), e o *Function Block* coloca a *TRUE* a saída “*InTorque*” se o binário especificado é atingido. Este *Function Block* é aplicável para a força e para o binário. Quando não existe uma carga externa, a força é aplicável. O binário é positivo na direção positiva da velocidade.

Na figura 3.24 podemos ver como está constituído o FB.

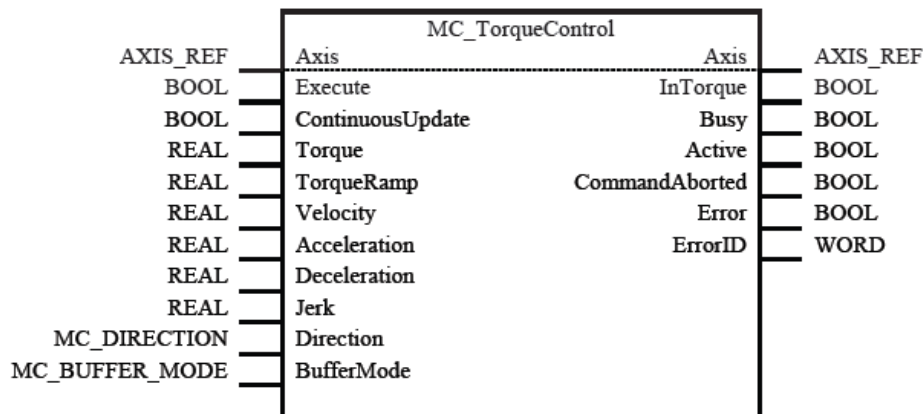


Figura 3.24: MC_TorqueControl FB [5]

Entradas do FB:

- AXIS_REF
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *Torque* - Valor do binário;
- *TorqueRamp* - Máxima derivada em ordem ao tempo do valor do binário ou da força;
- *Velocity* - Valor máximo que a velocidade pode atingir;
- *Acceleration* - Entrada opcional, indica o valor da aceleração, é sempre positivo;
- *Deceleration* - Entrada opcional, que indica o valor da desaceleração;
- *Jerk* - Esta é uma entrada opcional que diz qual o valor pretendido para o *Jerk* que é a variação de aceleração em função do tempo.
- *Direction* - Pode ter 1 de 2 valores: *mcPositiveDirection*, *mcNegativeDirection*;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- AXIS_REF
- *InTorque* - Vai a *TRUE*, quando é atingido o binário/força especificados;

- *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_PositionProfile** - Este é um *Function Block* que comanda um perfil de movimento da posição em função do tempo.

Na figura 3.25, mostra quais as entradas e saídas do FB.

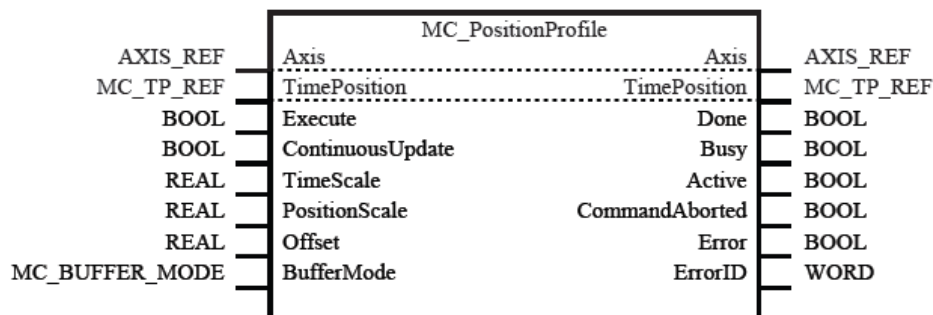


Figura 3.25: MC_PositionProfile FB [5]

Entradas do FB:

- *AXIS_REF*
- *TimePosition* - Referência para o Tempo/Posição. Este par pode também ser expresso como *Deltatempo/Posição*, onde *Delta* é a diferença entre dois pontos consecutivos;
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *TimeScale* - Entrada opcional, que representa o fator de escala do tempo do perfil;
- *PositionScale* - Entrada opcional, que representa o fator de escala da posição;
- *Offset* - Entrada opcional, indica qual o offset do perfil;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF*
 - *Time_Position*
 - *Done* - Fica a *TRUE*, quando o perfil está completo;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_VelocityProfile** - Este FB comanda um perfil de movimento da velocidade em função do tempo. A velocidade do elemento final presente no perfil é mantida.

A constituição do FB está presente na figura 3.26.

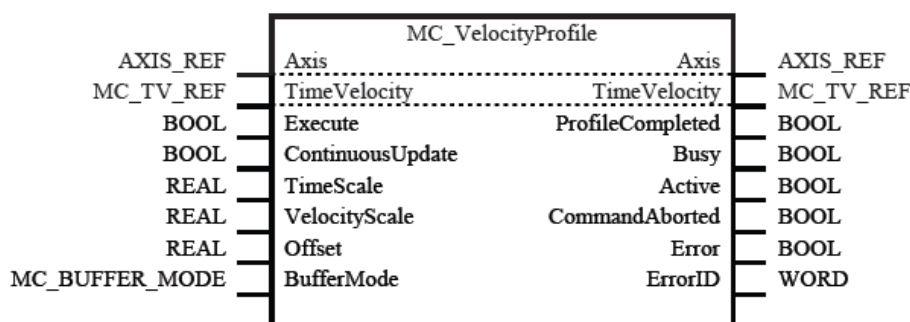


Figura 3.26: MC_VelocityProfile FB [5]

Entradas do FB:

- *AXIS_REF*
- *TimeVelocity* - Referência para o Tempo/Velocidade. Este par pode também ser expresso como *Deltatempo/Velocidade*, onde *Delta* é a diferença entre dois pontos consecutivos;
- *Execute* - Quando houver um *rising edge* inicia o movimento;
- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *TimeScale* - Entrada opcional, que representa o fator de escala do tempo do perfil;
- *VelocityScale* - Entrada opcional, que representa o fator de escala da velocidade;

- *Offset* - Entrada opcional, indica qual o offset do perfil;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF*
 - *Time_Velocity*
 - *ProfileCompleted* - Fica a *TRUE*, quando o perfil está completo;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_AccelerationProfile** - Tal como os FB anteriores, este comanda um perfil de movimento sendo que neste caso se trata de um perfil de aceleração. Este *Function Block* quando finaliza o perfil mantém a velocidade final e a aceleração vai para zero.

A constituição do FB pode ser vista na figura 3.27.

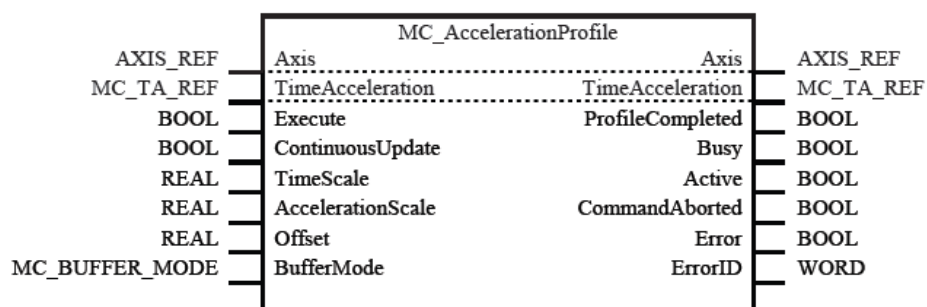


Figura 3.27: MC_AccelerationProfile FB [5]

Entradas do FB:

- *AXIS_REF*
- *TimeAcceleration* - Referencia para o Tempo/Aceleração. Também pode ser expresso por *Deltatempo/Aceleração*, onde o *Delta* é a diferença entre dois pontos consecutivos;
- *Execute* - Quando houver um *rising edge* inicia o movimento;

- *ContinuousUpdate* - Entrada opcional, que quando o FB for acionado, e se esta estiver a *TRUE*, vai fazer com que o FB use os valores atuais das variáveis de entrada e aplicá-los ao movimento em curso;
- *TimeScale* - Entrada opcional, que representa o fator de escala do tempo do perfil;
- *AccelerationScale* - Entrada opcional, que representa o fator de escala para a amplitude da aceleração;
- *Offset* - Entrada opcional, indica qual o offset do perfil;
- *BufferMode* - Esta é uma entrada opcional que define a sequência cronológica do FB de acordo com os modos disponíveis, ver subcapítulo 3.4.2.3.

Saídas do FB:

- *AXIS_REF*
 - *Time_Acceleration*
 - *ProfileCompleted* - Fica a *TRUE*, quando o perfil está completo;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Active* - Saída opcional, que indica que o FB tem controlo sobre o eixo;
 - *CommandAborted* - Saída opcional, que diz que o comando que estava a ser executado foi abortado por outro;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_SetPosition** - Este *Function Block* muda o sistema de coordenadas de um eixo, manipulando tanto a posição do ponto de ajuste, bem como a posição real de um eixo com o mesmo valor, sem causar qualquer movimento (Re-calibração com mesmo erro). Isto pode ser utilizado, por exemplo, para uma situação de referência. Este bloco de função também pode ser usado durante o movimento, sem alterar a posição comandada, que agora está posicionada no sistema de coordenadas deslocado.

Na figura 3.28, apresenta as entradas e saídas do FB.

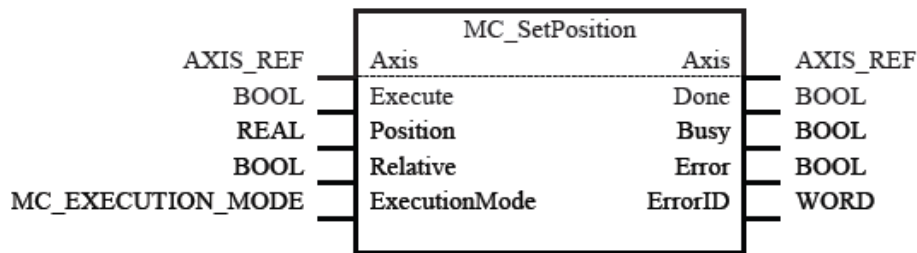


Figura 3.28: MC_SetPosition FB [5]

Entradas do FB:

- **AXIS_REF**
- *Execute* - Quando for *TRUE*, começa a ajustar a posição;
- *Position* - Unidade de posição, significa "*Distance if Relative = TRUE*";
- *Relative* - Entrada opcional, indica a distância relativa e significa "*if TRUE, 'Absolute' position, if False = Default*";
- *ExecutionMode* - Esta é uma entrada opcional e define a sequência do FB: *mcImmediately*, em que a funcionalidade é validada imediatamente podendo influenciar o movimento que está a ocorrer, o outro modo é o *mcQueued* - mesma funcionalidade que o *buffer mode Buffered*.

Saídas do FB:

- **AXIS_REF**
 - *Done* - Fica a *TRUE*, quando a entrada "Position" tem um novo valor;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_ReadActualPosition** - Este FB retorna a posição atual do eixo

A sua constituição pode ser vista na figura 3.29.

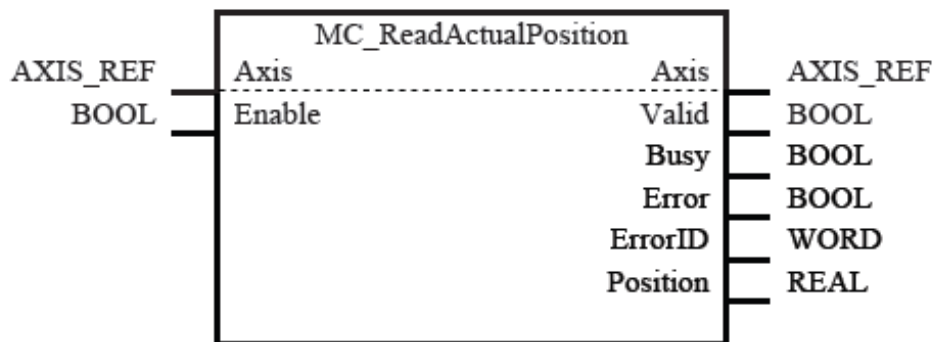


Figura 3.29: MC_ReadActualPosition FB [5]

Entradas do FB:

- *Axis* - Referência do eixo.
- *Enable* - Enquanto estiver ativo, o FB está continuamente a obter o valor da posição.

Saídas do FB:

- *Valid* - Fica a *TRUE*, quando uma saída válida está disponível no FB;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
 - *Position* - Valor atual da Posição.
- **MC_ReadActualVelocity** - Este *Function Block* retorna o valor da velocidade atual enquanto o *Enable* estiver ativo. A saída *Valid* é *TRUE* quando a saída *Velocity* é válida. Se for feito um *reset* ao *Enable*, os dados perdem a sua validade. O *reset* é feito a todas as saídas mesmo que haja novos dados disponíveis.

Na figura 3.30 podemos ver a entradas e saídas constituintes do FB.

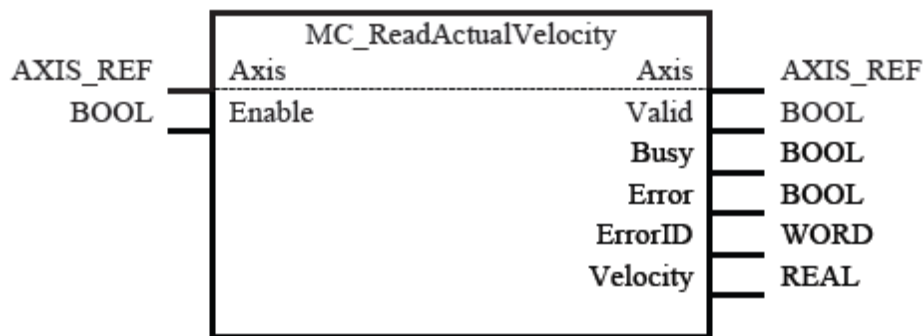


Figura 3.30: MC_ReadActualVelocity FB [5]

Entradas do FB:

- **AXIS_REF**
- *Enable* - Enquanto estiver ativo, o FB está continuamente a obter o valor da velocidade.

Saídas do FB:

- **AXIS_REF**
 - *Valid* - Fica a *TRUE*, quando uma saída válida está disponível no FB;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
 - *Velocity* - Valor atual da Velocidade.
- **MC_ReadActualTorque** - É um FB que retorna o valor do binário atual enquanto o *Enable* estiver ativo. A saída *Valid* é *TRUE* quando a saída *Torque* é válida. Se for feito um *reset* ao *Enable* os dados perdem a sua validade. O *reset* é feito a todas as saídas mesmo que haja novos dados disponíveis.

A constituição do FB está presente na figura 3.31.

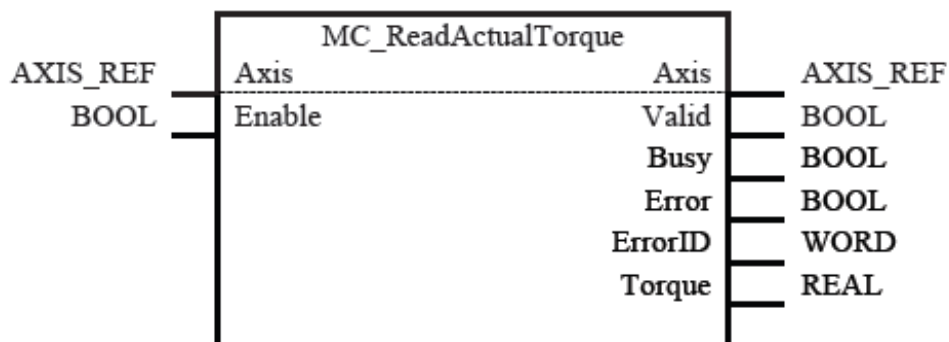


Figura 3.31: MC_ReadActualTorque FB [5]

Entradas do FB:

- *Axis* - Referência do eixo.
- *Enable* - Enquanto estiver ativo, o FB está continuamente a obter o valor do binário.

Saídas do FB:

- *Axis* - Referência do eixo.
 - *Valid* - Fica a *TRUE*, quando uma saída válida está disponível no FB;
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
 - *Torque* - Valor atual do Binário.
- **MC_Reset** - Este *Function Block* faz a transição do estado “*ErrorStop*” para o estado “*StandStill*” ou “*Disabled*”, realizando um *reset* a todos os erros relacionados com o eixo interno. Este FB não afeta a saída das instâncias dos outros FB.

As entradas e saídas do *Function Block* podem ser vistas na figura 3.32.

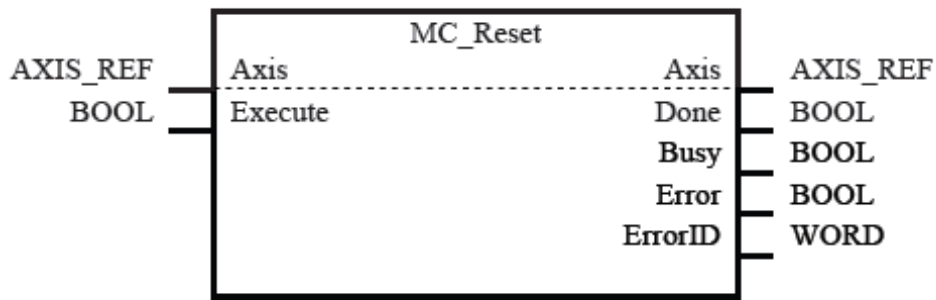


Figura 3.32: MC_Reset FB [5]

Entradas do FB:

- **AXIS_REF**
- *Enable* - Quando for *TRUE*, realiza o *reset* a todos os erros relacionados com o eixo;

Saídas do FB:

- **AXIS_REF**
 - *Done* - Fica a *TRUE*, quando o eixo atinge o estado "*StandStill*" ou "*Disable*";
 - *Busy* - Saída opcional que, enquanto for *TRUE*, indica que o FB ainda não terminou a sua tarefa;
 - *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
 - *ErrorID* - Saída opcional, identificação do erro.
- **MC_DigitalCamSwitch** - Este *Function Block* é uma analogia às válvulas no veio de um motor: comanda um conjunto de bits de saída discretos para mudar analogamente um conjunto de comutadores mecânicos ligados ao eixo. Também é possível realizar movimentos para a frente e para trás.

Na figura 3.33 mostra a constituição do FB.

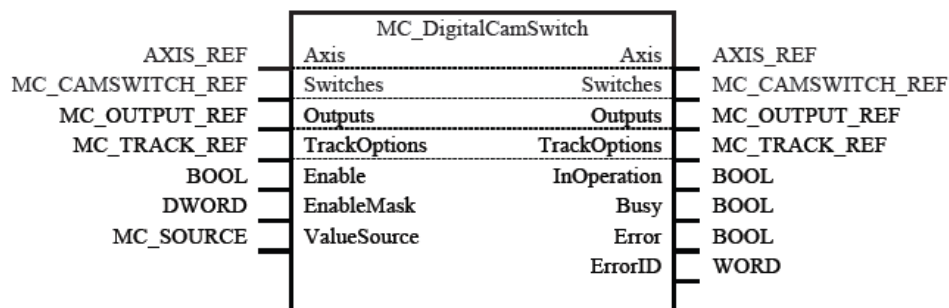


Figura 3.33: MC_DigitalCamSwitch FB [5]

Entradas do FB:

- AXIS_REF
- MC_CAMSWITCH_REF - referência das ações de mudança dos comutadores;
- MC_OUTPUT_REF - esta entrada/saída opcional é a referência para os sinais das saídas diretamente relacionados para as faixas referenciadas (max. 32 por FB, *First output = first = TrackNumber*);
- MC_TRACK_REF - entrada/saída opcional que é a referência para a estrutura que contém propriedades da faixa relacionada, por exemplo, as compensações ON e OFF por saída/faixa.
- *Enable* - Ativa as saídas "Switch";
- *EnableMask* - BOOL de 32 bits. Ativa as diferentes faixas. O bit menos significativo é correspondente ao *TrackNumber* mais baixo. Quando um dos bits for a 1 o *TrackNumber* correspondente a esse bit é ativado. Esta é uma entrada opcional.
- *ValueSource* - Entrada opcional que define a origem dos valores do eixo (e.g. posições), temos duas opções: *mcSetValue* que faz a sincronização no valor definido e o *mcActualValue* que realiza a sincronização no valor atual.

Saídas do FB:

- AXIS_REF
- MC_CAMSWITCH_REF
- MC_OUTPUT_REF
- MC_TRACK_REF
- *InOperation* - é TRUE quando as faixas comandadas estão ativas.
- *Busy* - Saída opcional que, enquanto for TRUE, indica que o FB ainda não terminou a sua tarefa;

- *Error* - Vai a 1, se algum erro acontecer no *Function Block*;
- *ErrorID* - Saída opcional, identificação do erro.

Os *Function Blocks* que foram apresentados podem ser divididos em FB fundamentais, isto é, FB que são necessários para que seja possível criar uma interface em que exista movimento, ou seja, controlo de um motor. Os *Function Blocks* chamados de fundamentais são os seguintes:

- MC_Power;
- MC_Home;
- MC_Stop;
- MC_Halt;
- MC_MoveAbsolute;
- MC_MoveRelative;
- MC_TorqueControl;
- MC_Reset;

Podemos ainda ter outro grupo, que é o dos FB secundários. Estes são FB que podem ser feitos a partir dos FB fundamentais, bem como FB que são independentes do hardware ou FB que retornam o estado do sistema. Dos seleccionados os FB secundários são:

- MC_MoveContinuousAbsolute - MC_MoveVelocity + MC_MoveAbsolute;
- MC_MoveContinuousRelative - MC_MoveVelocity + MC_MoveRelative;
- MC_ReadActualPosition;
- MC_ReadActualVelocity;
- MC_ReadActualTorque;
- MC_PositionProfile - independente do *hardware*;
- MC_VelocityProfile - independente do *hardware*;
- MC_AccelerationProfile - independente do *hardware*;
- MC_DigitalCamSwitch;
- MC_MoveSuperimposed - MC_MoveAbsolute + MC_MoveRelative + MC_Velocity;
- MC_HaltSuperimposed;

Capítulo 4

Desenvolvimento

Neste capítulo apresenta-se o trabalho desenvolvido para o movimento do motor. De seguida, descreve-se com pormenor o protocolo RS232 desenvolvido até aos testes finais realizados no Beremiz.

4.1 Protocolo de Comunicação

O objetivo deste trabalho é provar que os *Motion Control Function Blocks* criados pela PL-COpen conseguem realizar a interligação entre os PLC's e as máquinas CNC, visto que estas não podem ser programadas usando as linguagens de programação presentes na norma IEC 61131-3. Para isso, foi escolhido o eixo presente no laboratório I105 da Faculdade de Engenharia da Universidade do Porto, para servir como plataforma de validação dos mesmos, mas para que fosse possível a comunicação entre o ambiente de desenvolvimento Beremiz e o motor PD4-N era necessário um protocolo de comunicação. O motor da Nanotec funciona com dois protocolos possíveis, o CANOpen e o protocolo de comunicação série.

Devido ao facto de o CANOpen não funcionar muito bem no Beremiz, o protocolo escolhido foi o RS232, mas antes de iniciar o desenvolvimento do protocolo foi necessário arranjar um conversor RS232/RS485, pois o motor PD4-N funciona com o protocolo RS485 e o computador no qual o protocolo e o Beremiz vão estar usa RS232.

Após todas as ligações estarem feitas, foi necessário verificar se o *hardware*, neste caso o motor, estava a funcionar corretamente e a comunicar com o computador. Para este teste foi usado o programa fornecido pela Nanotec, o NanoPro. Para a realização correta do teste foram seguidos os passos disponíveis no manual do motor PD4-N.

Na tabela 4.1, são apresentados os passos que foram efetuados:

Tabela 4.1: Uso do *software* NanoPro [1]

Step	Action
1	Install the NanoPro control software on your PC. See the NanoPro separate manual.
2	Connect the PC to the RS485 interface of the Plug & Drive motor according to the connection diagram.
3	Switch on the operating voltage (12-48 V DC). If operating voltage > 50 V, the output stage is destroyed.
4	If necessary, install the driver for the converter cable ZK-RS485-USB.
5	Connect the Plug & Drive motor to the PC.
6	Start NanoPro software.
7	Select the Communication tab
8	In the "Port"field select the COM port to which the PD4-N is connected
9	Select the "115200 bps"entry in the "Baudrate"selection field
10	Check the current setting using the motor data sheet on the "Motor Settings"tab. Presettings: Phase Current equal to 50%(current level) and Phase current during idle equal to 25%.
11	Select the "Movement Mode"tab
12	Click on the "Test Record"button to carry out the pre-set travel profile.
13	Now we can enter the required settings or create a new travel profile.

Para uma melhor visualização do ambiente do programa, tomemos atenção à figura 4.1:

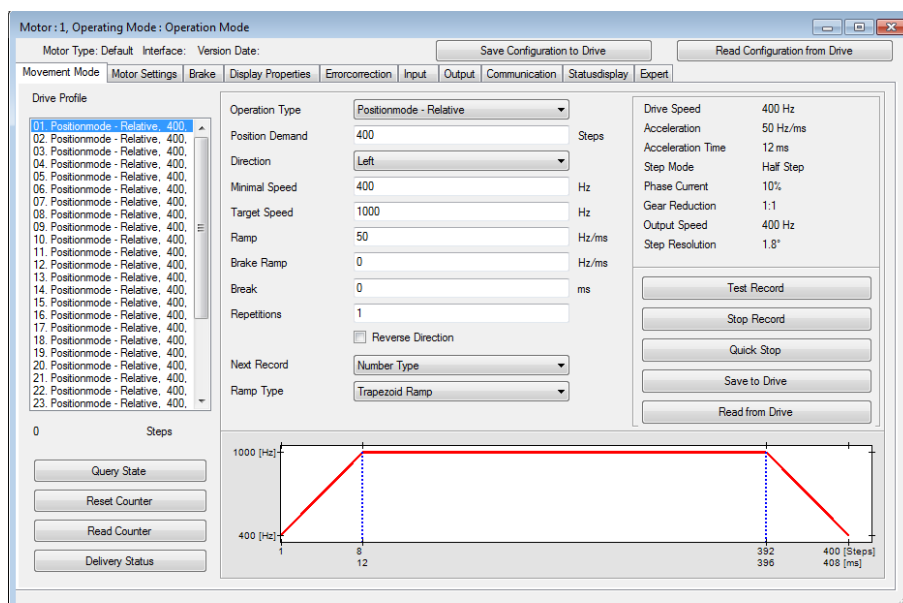


Figura 4.1: Movement Tab do NanoPro [1]

Depois de se verificar que o motor comunicava e funcionava corretamente, deu-se início ao desenvolvimento do protocolo RS232.

Para conseguirmos fazer com que o motor se movimente é necessário enviar para este o comando desejado, recorrendo-se ao *Programming Manual* fornecido pela Nanotec.

Um comando inicia-se com o caracter # e termina com o *carriage return* '\r'. O caracter inicial é seguido por um endereço do motor representado como um número decimal em ASCII. O valor desse número pode estar desde 1 até 254, se '*' é enviado em vez de um número, todos os *drivers* conectados ao barramento são endereçados [11]. O endereço do motor é seguido pelo comando em si, que geralmente consiste num caracter em ASCII ou um número ASCII opcional. É necessário ter em atenção que alguns comandos consistem em vários caracteres enquanto que outros não requerem um número, mas sim letras, como um parâmetro.

No parágrafo anterior falamos da estrutura do comando enviado, agora vamos falar da resposta que confirma a receção do comando. Após o envio de um comando, se o controlador o reconhecer, este confirma a sua receção retornando o comando como um *echo*, mas sem o caracter inicial #. No caso de o controlador receber um comando que não conheça, este responde enviando o comando seguido de um ponto de interrogação '?'. A resposta do controlador tal como o comando que é enviado terminam com o *carriage return* '\r'. No caso de serem enviados valores inválidos ao controlador, estes são ignorados mas enviados na mesma como um *echo* [11]. De seguida são apresentados alguns exemplos do envio do comando e da sua receção.

- Valor transmitido ao controlador: '#1u1000\r'
- Resposta do *firmware*: '1u1000\r'
- Especificar a distancia ao controlador 1: '#1s1000\r' -> '1s1000\r'
- Iniciar o motor: '#1A\r' -> '1A\r'
- Comando inválido: '#1^\r' -> '1^\r'

Agora vai ser apresentado o protocolo desenvolvido falando um pouco do objetivo de cada função.

Antes de mais, é necessário mencionar o uso da biblioteca "serial_util" que possui as funções necessárias para a realização da comunicação série entre o computador e o motor.

As funções presentes são as seguintes:

- serial_open - função que abre o canal de comunicação série;
- serial_close - fecha o canal de comunicação;
- serial_config - realiza a configuração do canal de comunicação;
- serial_read - permite ler o que está presente no canal de comunicação, neste caso ler a resposta do controlador;

- **serial_write** - função que permite escrever no canal de comunicação, no presente caso o envio de comandos para o motor.

Agora vão ser apresentadas as funções desenvolvidas, sendo dada uma breve explicação da sua utilidade, bem como o seu cabeçalho e o comando enviado.

- **serial_device** - esta função realiza a abertura do canal de comunicação.

Cabeçalho - `int serial_device(char *serport)`

A variável *serport* corresponde à porta série que vai ser usada, visto que apenas é usado um motor, a porta `ttyS0` foi a única escolhida.

- **currentPos** - tem como objetivo indicar qual a posição em que está o motor.

Cabeçalho - `void currentPos(int serfd)`

A variável *serfd* corresponde ao *file descriptor* da ligação série. Esta variável é necessária em todas as funções, exceto na *serial_device* pois é aí que este é criado.

Comando enviado - `'#1Cr'`.

- **step** - permite especificar o tipo de *step* pretendido. O valor escolhido é o equivalente ao número de *microsteps* por *full step*, com exceção do valor 254 que seleciona o *feed rate mode* e do valor 255 que escolhe o *adaptive step mode*. Os valores permitidos são: 1, 2, 4, 5, 8, 10, 16, 32, 64, 254 e 255.

Cabeçalho - `void step(int serfd, int stepvalue)`

Como o nome indica a variável *stepvalue* armazena o valor desejado para o *step*.

Comando enviado - `'#1gxxxx\r'`, onde *xxxx* é o valor proveniente da variável *sentstep*.

- **MotorStart** - função que inicia o motor com os parâmetros escolhidos.

Cabeçalho - `void MotorStart(int serfd)`

Comando enviado - `'#1A\r'`.

- **rampType** - seleciona o tipo de rampa para todos os modos, este pode ter 1 de 3 valores possíveis:

- 0 - Rampa Trapezoidal;
- 1 - Rampa Sinusoidal;
- 2 - Rampa *jerk-free*

Cabeçalho - `void rampType(int serfd, int ramp)`

A variável *ramp* guarda o valor escolhido pelo utilizador para a seleção da rampa, valor este que pode ter um dos três valores indicados anteriormente.

Comando enviado - `'#1:ramp_mode=n\r'`, *n* corresponde ao número guardado na variável *ramp*.

- **MotorStop** - esta função para o movimento que está a ser executado.

Cabeçalho - `void MotorStop(int serfd)`

Comando enviado - `'#1Sr'`.

- **PositioningMode** - esta função tem como objetivo escolher que modo desejamos para o deslocamento do motor. Os modos disponíveis são os seguintes:

- 1 - *Relative Positioning*
- 2 - *Absolute Positioning*

Cabeçalho - void PositioningMode(int serfd, int mode)

A variável *mode* pode ter 1 de 2 valores, se possuir o valor 1 o motor deslocar-se-á no modo relativo caso possua o valor 2 este usará o modo de deslocamento absoluto.

Comando enviado - '#1px\r', onde x corresponde ao número escolhido pelo utilizador de acordo com o tipo de deslocamento que deseja usar.

- **TravelDistance** - como o nome indica esta função serve para especificar a distancia/posição que queremos que seja percorrida, isto é, no caso de ser escolhido o deslocamento relativo este comando especifica a distância a ser percorrida, neste caso apenas valores positivos são permitidos e a direção do movimento tem que ser escolhida via um comando presente na função rotationDirection que irá ser apresentado brevemente. No caso de ser escolhido o movimento absoluto este comando especifica a posição para onde se quer ir, relativamente à direção do movimento já não é necessária a função rotationDirection, pois neste modo valores negativos já são aceites, ou seja, se o valor for positivo este movimenta-se para a direita, se for negativo movimenta-se para a esquerda.

Cabeçalho - void TravelDistance(int serfd, int distance)

Na variável *distance* é onde está guardado o valor escolhido.

Comando enviado - '#1sxxxx\r', onde xxxx corresponde ao valor presente na variável *distance*.

- **MinimumFreq** - esta função especifica a velocidade mínima, em Hertz, a que o motor pode andar. Aceita valores desde 1 até 160000.

Cabeçalho - void MinimumFreq(int serfd, int initfreq)

A variável *initfreq* armazena o valor escolhido para a velocidade.

Comando enviado - '#1uxxxx\r', onde xxxx corresponde ao valor guardado em *initfreq*.

- **MaximumFreq** - esta função especifica a velocidade máxima, em Hertz, a que o motor pode andar. A velocidade máxima é atingida depois de passar pela rampa de aceleração.

Cabeçalho - void MaximumFreq(int serfd, int maxifreq)

A variável *maxifreq* armazena o valor escolhido para a velocidade.

Comando enviado - '#1oxxxx\r', onde xxxx corresponde ao valor guardado em *maxifreq*.

- **AccelRamp** - especifica qual o valor para a rampa de aceleração.

Cabeçalho - void AccelRamp(int serfd, int num)

Comando enviado - '#1bxxxx\r', onde xxxx corresponde ao valor guardado na variável *num*.

- **BrakeRamp** - especifica qual o valor para a rampa de aceleração. Se o valor especificado for 0 significa que o valor usado na rampa de aceleração é usado também na rampa de travagem.

Cabeçalho - void BrakeRamp(int serfd, int value)

Comando enviado - '#1Bxxxx\r', onde xxxx corresponde ao valor guardado na variável *value*.

- **rotationDirection** - Tal como o nome indica esta função tem como objetivo definir o sentido da rotação. Não esquecer que esta função apenas funciona se o modo de deslocamento for o modo relativo. Para escolher a direção no comando a enviar, tem que ir um dos seguintes valores:

- 0 - Esquerda;
- 1 - Direita;

Cabeçalho - void rotationDirection(int serfd, int direction)

Na variável *direction* é onde vai o valor que especifica qual a direção a tomar.

Comando enviado - '#1dx\r', onde x corresponde ao valor proveniente da variável *distance*.

- **NumberRepetitions** - esta função consiste em definir um número de repetições para o movimento que está a ser definido.

Cabeçalho - void NumberRepetitions(int serfd, int number)

A variável *number* armazena o número de repetições escolhido.

Comando enviado - '#1Wxxx\r', onde xxx corresponde ao número de repetições armazenado na variável *number*.

- **repetitionDirection** - esta função define se deve ser alterada a direção da rotação após cada repetição, esta função apenas aceita dois valores:

- 0 - Não haver mudança de direção;
- 1 - Para existir mudança de direção

Cabeçalho - void repetitionsDirection(int serfd, int change)

Comando enviado - '#1tx\r', onde x corresponde a um dos números mencionados em cima e que estão guardados na variável *change*.

- **AccelJerk** - Especifica o *jerk* máximo para a aceleração.

Cabeçalho - void AccelJerk(int serfd, int acc)

Comando enviado - '#1:bx\r', onde x corresponde ao valor do *jerk* escolhido e guardado na variável *acc*.

- **BrakeJerk** - Indica qual o máximo *jerk* para a rampa de travagem. Se o valor escolhido for 0, o valor usado para a travagem será o mesmo que é usado para a aceleração.

Cabeçalho - void BrakeJerk(int serfd, int brk)

Comando enviado - '#1:Bx\r', onde x corresponde ao valor do *jerk* escolhido para a rampa de travagem.

Depois de desenvolvido, foram realizados testes ao protocolo de forma a se verificar se cada função realizava o seu objetivo. Verificou-se que a cada envio de um comando o controlador respondia da maneira esperada, maneira esta que é um *echo* do comando enviado tal como foi mencionado anteriormente. Também foi verificado se este realizava o movimento pretendido, isto é, foi colocado um valor para a distância no caso de um movimento relativo e um para a posição no caso de um movimento absoluto. Após o deslocamento foi realizada uma medição verificando-se se o motor tinha andado a distância pretendida e no outro modo se este tinha se deslocado para a posição desejada. Este teste foi realizado para diferentes valores de distância e posição concluindo-se que o protocolo desenvolvido funcionava corretamente.

4.2 Biblioteca MCFunctionBlocks

Após a conclusão do protocolo era necessário desenvolver no Beremiz os *Function Blocks* pretendidos. Para isso foi sugerido pelo orientador realizar o seu desenvolvimento através de um ficheiro *XML*, criando uma biblioteca no ambiente de desenvolvimento com os *Motion Control Functions Blocks* que aparece na janela *Library* do Beremiz.

Na figura 4.2, mostra a aparência do ficheiro bem como o seu cabeçalho inicial:

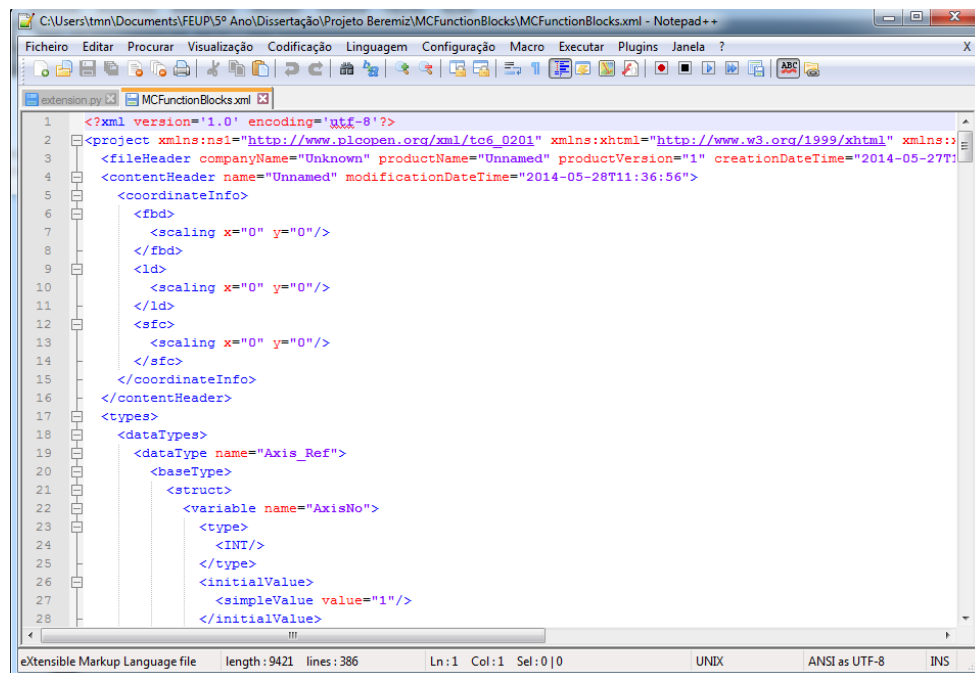


Figura 4.2: Ficheiro XML

Antes de iniciar a construção dos FB, foi necessário criar um tipo de dados especial o `AXIS_REF` que é uma estrutura que contém informação sobre o eixo em funcionamento. Como já foi dito anteriormente este é usado como uma `VAR_IN_OUT` em todos os *Motion Control Function Blocks*. Quanto conteúdo desta estrutura, este é dependente da implementação desenvolvida e em última instância pode estar vazia. Na figura 4.3 mostra um exemplo de como pode estar definido este tipo de dados.

```

AXIS_REF data type declaration:
TYPE
  AXIS_REF : STRUCT
    (Content is implementation dependent)
  END_STRUCT
END_TYPE

Example:
TYPE
  AXIS_REF : STRUCT
    AxisNo: UINT;
    AxisName: STRING (255);
    ...
  END_STRUCT
END_TYPE

```

Figura 4.3: Exemplo do tipo de dados `AXIS_REF`

No caso desta dissertação, esta estrutura contém apenas a referência para o motor em uso, tal pode ser verificado na figura 4.4.

```

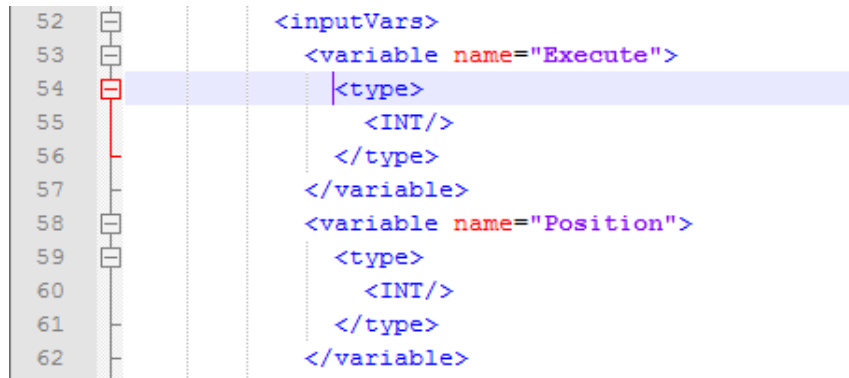
17 <types>
18   <dataTypes>
19     <dataType name="Axis_Ref">
20       <baseType>
21         <struct>
22           <variable name="AxisNo">
23             <type>
24               <INT/>
25             </type>
26             <initialValue>
27               <simpleValue value="1"/>
28             </initialValue>
29           </variable>

```

Figura 4.4: Estrutura `AXIS_REF`

Após a definição deste tipo de dados, prosseguiu-se para o desenvolvimento do POU, neste caso os *Function Blocks*. Primeiramente é necessário indicar quais as entradas e saídas presentes no FB, na figura 4.5 pode-se ver como são declaradas as entradas do FB.

É de notar que as variáveis que são do tipo BOOL estão com o tipo INT, isto porque na linguagem C o tipo BOOL não pode ser usado, pelo que é usado o tipo INT para representar este tipo quando necessário.



```

52  <inputVars>
53  <variable name="Execute">
54  <type>
55  <INT/>
56  </type>
57  </variable>
58  <variable name="Position">
59  <type>
60  <INT/>
61  </type>
62  </variable>

```

Figura 4.5: Declaração das entradas dos FB

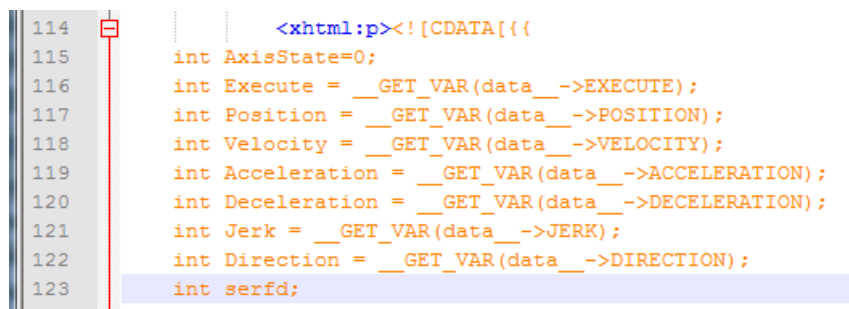
No caso das saídas e das variáveis entrada/saída no local onde aparece *inputVars*, coloca-se *outputVars* e *InOutVars*, respetivamente.

Após a definição de todas as variáveis, passa-se para o desenvolvimento do corpo do FB. Este foi feito usando a linguagem C, isto é possível devido ao compilador Matiec, pois o que este faz é converter a linguagem em que está escrito o projeto IEC 61131-3 para linguagem C, pelo que é permitido a escrita em C.

Falando mais propriamente do conteúdo do corpo dos FB, o código no interior FB consiste na chamada das funções criadas no protocolo criado e apresentado no subcapítulo anterior.

De seguida, será apresentado o conteúdo do FB MC_MoveRelative de forma a se perceber como foi efetuado o desenvolvimento do código do FB.

Para se conseguir aceder às entradas do FB, é necessário usar a função `__GET_VAR(data__->nomeVariável)`, na imagem 4.6 podemos ver como foi feita essa associação.



```

114  <xml:p><![CDATA[{{
115  int AxisState=0;
116  int Execute = __GET_VAR(data__->EXECUTE);
117  int Position = __GET_VAR(data__->POSITION);
118  int Velocity = __GET_VAR(data__->VELOCITY);
119  int Acceleration = __GET_VAR(data__->ACCELERATION);
120  int Deceleration = __GET_VAR(data__->DECELERATION);
121  int Jerk = __GET_VAR(data__->JERK);
122  int Direction = __GET_VAR(data__->DIRECTION);
123  int serfd;

```

Figura 4.6: Associação das entradas do MC_MoveRelative

Após esta associação, os valores colocados nas entradas do FB são colocados nas variáveis criadas no corpo do FB.

De seguida, é necessário abrir o canal de comunicação, para evitar a constante abertura do canal foi criada uma variável auxiliar *AxisState* que passa a 1 quando o canal é aberto o que pode ser visto na figura 4.7. Quando o canal é fechado esta passa a 0.

```

124
125     if(AxisState == 0)
126     {
127         char port[] = "/dev/ttyS0";
128         serfd = serial_device(port);
129         AxisState == 1;
130     }
131

```

Figura 4.7: Abertura do canal de comunicação

Após este passo é necessário indicar quais as funções presentes no corpo do FB, isto é, colocar as funções que permitam que o FB comande o motor a realizar o movimento, tal pode ser observado na figura 4.8.

```

132     if (Execute == 1)
133     {
134         __SET_VAR(data__->,BUSY,__INT_LITERAL(1));
135         __SET_VAR(data__->,ACTIVE,__INT_LITERAL(1));
136         PositioningMode(serfd,1);
137         //printf(" %d\n",Position);
138         TravelDistance(serfd,Position);
139         MinimumFreq(serfd,400);
140         //printf(" %d\n",Velocity);
141         MaximumFreq(serfd,Velocity);
142         //printf(" %d\n",Acceleration);
143         AccelRamp(serfd,Acceleration);
144         //printf(" %d\n",Deceleration);
145         BrakeRamp(serfd,Deceleration);
146         //printf(" %d\n",Direction);
147         rotationDirection(serfd,Direction);
148         //printf(" %d\n",Jerk);
149         AccelJerk(serfd,Jerk);
150         BrakeJerk(serfd,0);
151         MotorStart(serfd);
152         __SET_VAR(data__->,DONE,__INT_LITERAL(1));
153         __SET_VAR(data__->,ERROR,__INT_LITERAL(0));
154     }
155
156     if(__GET_VAR(data__->DONE) == 1)
157     {
158         __SET_VAR(data__->,BUSY,__INT_LITERAL(0));
159         __SET_VAR(data__->,ACTIVE,__INT_LITERAL(0));
160     }
161     serial_close(serfd);
162     AxisState=0;

```

Figura 4.8: Funções usadas no MC_MoveRelative para comando do motor

O comando `__SER_VAR(data__->,nomeVariável,__type_LITERAL)` permite colocar a 1 ou a 0 as saídas do *Function Block*.

Quanto aos restantes FB, todos eles seguem estes passos, tendo apenas como diferenças as funções necessárias para o FB em questão.

Depois de desenvolvido o ficheiro *XML* que contém a biblioteca dos *MC_Function Blocks* é necessário colocá-la no Beremiz. Isto é feito através do ficheiro em *python*, chamado *extension.py* que realiza a criação da biblioteca no Beremiz, bem como a ligação da biblioteca com o protocolo desenvolvido de forma a que seja possível a comunicação entre o motor e o Beremiz. Na figura 4.9 pode-se ver como é a constituição do código.

```

1  import features
2  import os
3  from POULibrary import POULibrary
4
5  def GetLocalPath(filename):
6      return os.path.join(os.path.split(__file__)[0], filename)
7
8  class MCFunctionBlocks(POULibrary):
9
10     def GetLibraryPath(self):
11         return GetLocalPath("MCFunctionBlocks/MCFunctionBlocks.xml")
12
13     def Generate_C(self, buildpath, varlist, IECCFLAGS):
14         # minimal example in case you need to link to some shared library
15         # (here : lib math)
16         return ([], [], False), "/home/ee09195/BeremizMCFB/build/serial_util.c /home/ee09195/BeremizMCFB/bui
17
18     def GetMarioLibClass():
19         return MCFunctionBlocks
20
21 features.libraries.append(
22     ('MCFB', GetMarioLibClass))

```

Figura 4.9: *Extension.py*

As partes do código em que se define a biblioteca e quais os recursos que esta necessita, ou seja, para que o Beremiz saiba a localização da biblioteca e saber de onde vêm as funções usadas nos FB são:

- **def GetLibraryPath(self)** - que pode ser visto na linha 10 e que retorna o local onde se localiza o ficheiro *XML* onde estão desenvolvidos os *Function Blocks*, ou seja, a biblioteca criada;
- **def Generate_C(self, buildpath, varlist, IECCFLAGS)** - é o local onde se indica, se necessário, a localização de outras bibliotecas usadas no *XML*, neste caso o protocolo desenvolvido e a biblioteca *serial_util*.

Por fim, para que a biblioteca aparecesse no ambiente de desenvolvimento o Beremiz deveria ser iniciado pela linha de comandos da seguinte maneira:

"/Beremiz.py -e extension.py/MCFunctionBlocks"

Depois de concluída esta parte, passou-se para a construção de aplicações com os FB desenvolvidos e ao seu teste, isto vai ser apresentado com mais detalhe no subcapítulo seguinte.

4.3 MC_Function Blocks no Beremiz

Como mencionado no subcapítulo anterior o *XML* cria uma biblioteca com os FB desenvolvidos no Beremiz. Agora é necessário realizar aplicações com o que foi desenvolvido.

No Beremiz para criar uma aplicação é necessário criar um programa, tal é feito por carregar no '+' na janela *Project*. Após a criação basta arrastar para a janela de desenvolvimento o *Function Block* desejado.

Para que cada programa funcione é necessário ser criado uma tarefa e uma instância para esse programa senão este não irá compilar.

Neste subcapítulo vão ser mostrados, os FB testados e como foi feito esse teste.

O primeiro a ser testado foi o MC_MoveRelative. A sua constituição pode ser vista na figura 4.10.

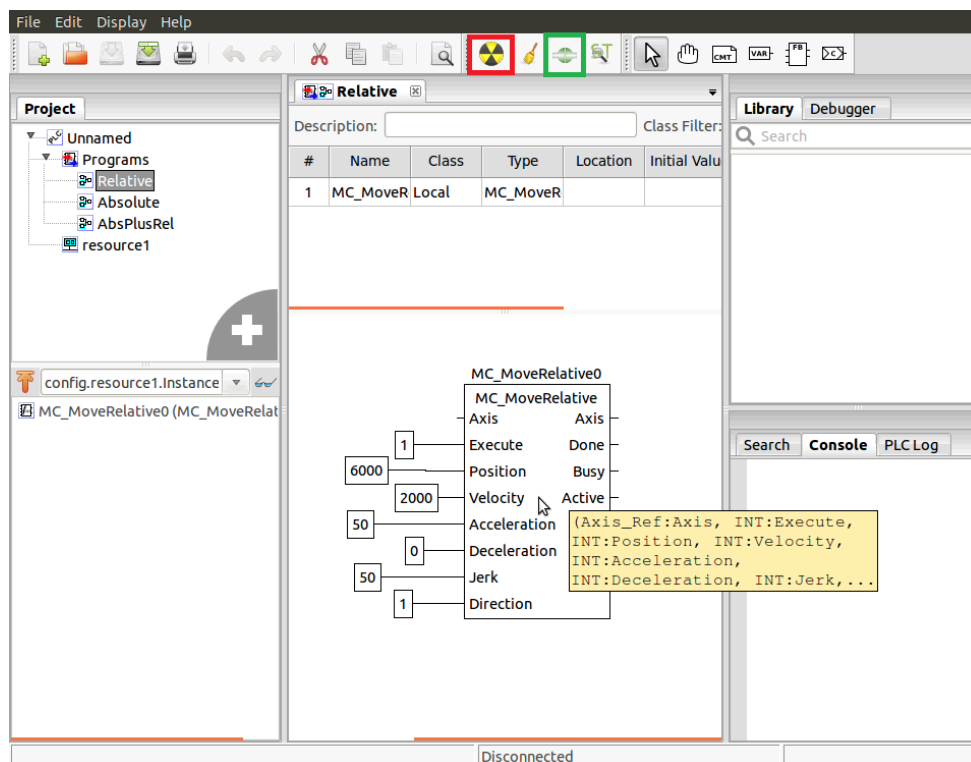


Figura 4.10: MC_MoveRelative no Beremiz

Após a configuração do FB com os valores desejados, carrega-se no botão de Compilar, botão assinalado a vermelho e se a compilação não assinalar nenhum erro realiza-se a conexão ao motor através do botão assinalado a verde. Pode-se verificar que o início do movimento do motor é um bocado demorado, isto é justificado pois o protocolo de comunicação usado entre o computador e o motor é o protocolo RS232.

Após concluir os passos anteriores, verificou-se a correta movimentação do motor, isto é, verificou-se que este andou a distância correta à velocidade escolhida. O teste realizado foi parecido com o usado no teste do protocolo, visto que o interior do FB é composto pelas várias funções

desenvolvidas no mesmo.

Depois do teste do MC_MoveRelative, testou-se o MC_MoveAbsolute verificando-se também o seu correto funcionamento, movimentando o motor para a posição desejada à velocidade especificada, na imagem 4.11 mostra a configuração deste *Function Block*, isto é, os valores constituintes das suas entradas.

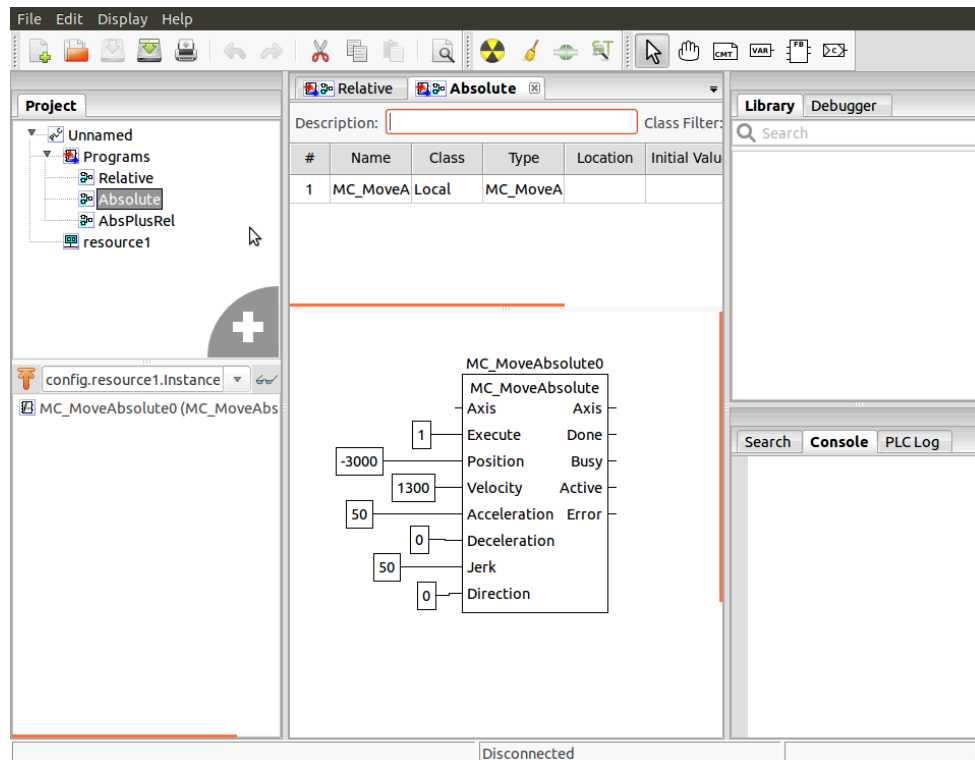


Figura 4.11: MC_MoveAbsolute no Beremiz

Após o teste, e depois de se verificar que ambos os FB funcionavam corretamente foi feito um programa em que estavam inseridos os dois *Function Blocks* de forma a se testar se a interligação entre eles funcionava nas perfeitas condições.

Para isso, foi conectado um MC_MoveAbsolute a um MC_Relative com valores diferentes de forma a se poder comparar se os valores se alteravam de FB para FB. Foram realizados testes com diferentes valores para a distância/posição e para velocidade, vendo também se o movimento mudava de direção quando tal era solicitado. Verificou-se também aquele tempo de espera entre cada *Function Block* como já mencionado, isto é devido ao protocolo RS232. Na figura 4.12 pode-se ver a interligação dos dois *Function Blocks*.

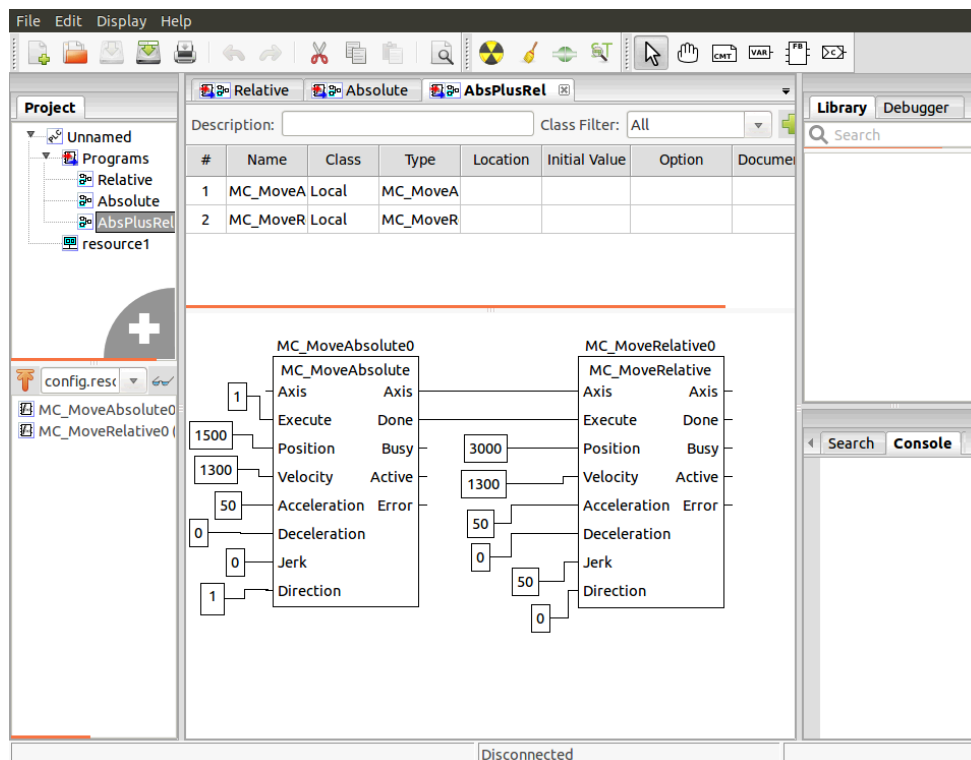


Figura 4.12: Interligação do FB MC_MoveAbsolute com o FB MC_MoveRelative

Foram ainda desenvolvidos os FB MC_Stop, que consiste tal como o seu nome indica parar o movimento que está a ser executado, e o MC_Power que inicia o motor.

Devido aos problemas de comunicação que foram acontecendo ao longo do desenvolvimento do projeto, desde da construção do protocolo até aos erros provenientes do Beremiz, o tempo foi escasseando, pelo que o orientador aconselhou a desenvolver estes quatro FB, e se o tempo o permitisse fazer mais algum, o que tal não foi possível.

Mas com o protocolo desenvolvido tal não seria muito complicado, ou seja, pelo que foi apresentado neste capítulo cada FB é implementado pelas funções presentes no protocolo, pelo que os outros FB básicos seriam implementados da mesma maneira. Quanto aos FB secundários, alguns deles podem ser desenvolvidos através dos FB básicos, como por exemplo o MC_MoveSuperimposed, que pode ser a interligação dos FB MC_MoveAbsolute, MC_MoveRelative e MC_Velocity.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões

Este trabalho tinha como objetivo o desenvolvimento dos *Motion Control Function Blocks*, desenvolvidos pela PLCOpen, no ambiente de desenvolvimento Beremiz, de forma a demonstrar que os FB podem ser usados como linguagem de programação dos PLC's para a interligação com as máquinas CNC, isto é, que seja possível a programação de máquinas CNC através das linguagens de programação presentes na norma IEC 61131-3.

O primeiro passo no projeto foi o estudo da norma IEC 61131-3, para se ficar com uma ideia de como esta estava organizada e quais os elementos presentes na norma, tais como, os *Program Organization Units* e as linguagens normalizadas por esta, tomando especial atenção aos *Function Blocks*.

Após o estudo da norma foi necessário realizar um breve estudo sobre o motor Nanotec PD4-N de forma a perceber quais as ligações. Após este estudo foram realizados testes de comunicação de forma a verificar se o motor funcionava corretamente e se a comunicação era feita sem problemas. Depois de verificado o *hardware* procedeu-se ao desenvolvimento de um protocolo em RS232, para isso, foi necessário recorrer ao *Programming Manual* do motor Nanotec de forma a se entender como eram enviados os comandos para o motor e como este confirmava a sua receção.

Concluído o desenvolvimento do protocolo, foi necessário realizar testes ao mesmo, onde ocorreram alguns problemas com a comunicação entre o computador e o motor, visto que este recebia os comandos mas não retornava o *echo* esperado a confirmar a receção do mesmo. Corrigido este problema conseguiu-se verificar o envio correto dos comandos e a sua receção por parte do motor. Verificou-se também que o motor se deslocava para as posições pretendidas e às velocidades impostas, sendo que isto foi verificado por vários testes em que foram medidos os deslocamentos.

Concluído com êxito o protocolo que faz a comunicação entre o Beremiz e o motor prosseguiu-se para a fase seguinte, o desenvolvimento dos *Motion Control Function Blocks*. Nesta fase, foi usada a linguagem *XML* onde foram desenvolvidos os FB de forma a que estes ficassem no Beremiz como uma biblioteca do mesmo. No ficheiro *XML*, foi necessário a declaração das entradas

de cada FB e o seu código foi feito em C recorrendo às funções criadas no protocolo.

Concluída esta parte prosseguiu-se para a etapa seguinte, o desenvolvimento de aplicações no Beremiz através dos FB criados. Ocorreram vários erros na compilação dos vários programas, bem como problemas relacionados com a comunicação entre o Beremiz e o motor, que fizeram com que esta fase demorasse mais que o esperado. Após resolvidos esses problemas, foram realizados vários testes de forma a se verificar se a função de cada FB estava a ser executada corretamente.

Mesmo ultrapassados todos os erros com sucesso, o tempo restante para o desenvolvimento de mais FB não era muito. Os problemas de comunicação e de compilação quer do *XML* quer do Beremiz não contribuíram para o desenvolvimento de todos os *Function Blocks*, pelo que apenas quatro foram desenvolvidos. Mas apesar de apenas estes terem sido feitos, foi possível concluir que com os *Motion Control Function Blocks* é possível controlar vários motores e realizar os movimentos desejados. Pelo que fica provado que através destes FB é possível a interligação entre PLC's e máquinas CNC de acordo com a norma 61131-3.

5.2 Trabalho Futuro

Inicialmente foi planeado o desenvolvimento de mais FB básicos e FB secundários, tal não foi possível devido aos inúmeros obstáculos. No futuro para melhorar este projeto seria necessário:

- Desenvolvimento de mais FB
- Realizar a comunicação com outro protocolo de comunicação, por exemplo CANOpen, de forma a que a comunicação seja mais rápida.

Anexo A

Ligações RS232

A.1 Ligações DTE-DCE e DTE-DTE

Nesta secção são apresentadas as ligações entre os dispositivos DTE e DCE, bem como entre dois DTE-DTE.

A figura A.1 mostra a ligação de um equipamento DTE a um equipamento DCE:

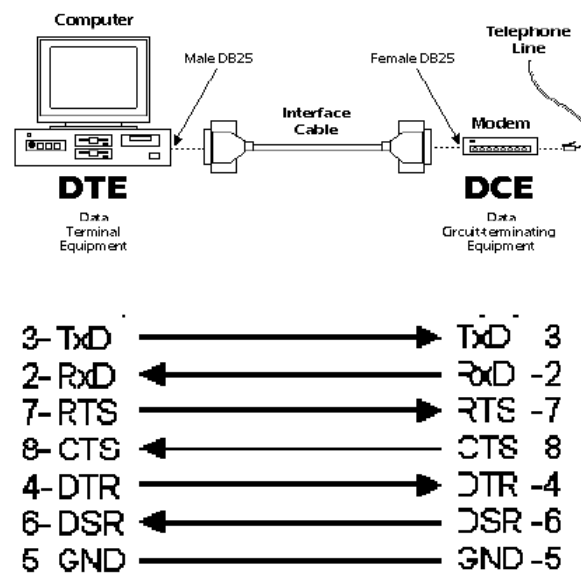


Figura A.1: Ligação DTE-DCE [6]

Na figura A.2, está representado outro tipo de ligação que é entre dois DTE:

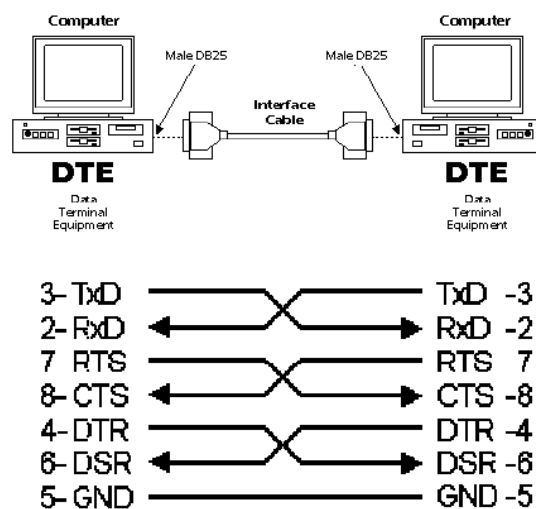


Figura A.2: Ligação DTE-DTE [6]

A.2 Sinais para os dispositivos DTE e DCE

A definição dos sinais para um dispositivo DTE é representada na figura A.3.

A negrito estão apresentados os sinais mais comuns.

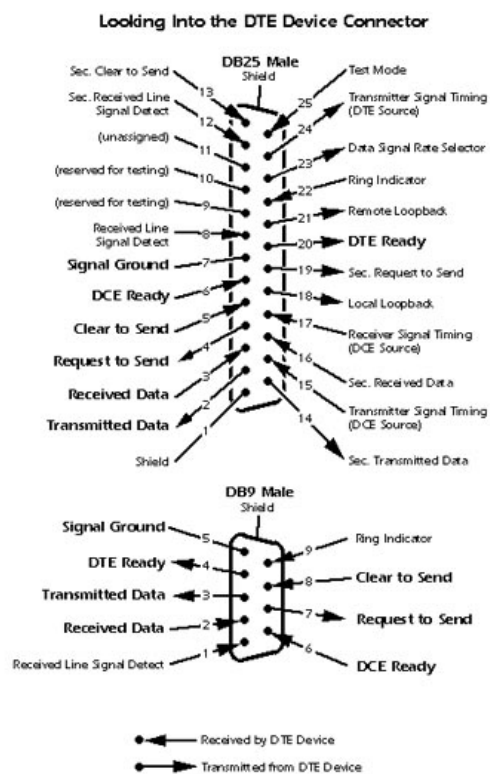


Figura A.3: Conector de um dispositivo DTE

Na figura A.4, está representado os sinais disponíveis para um dispositivo DCE:

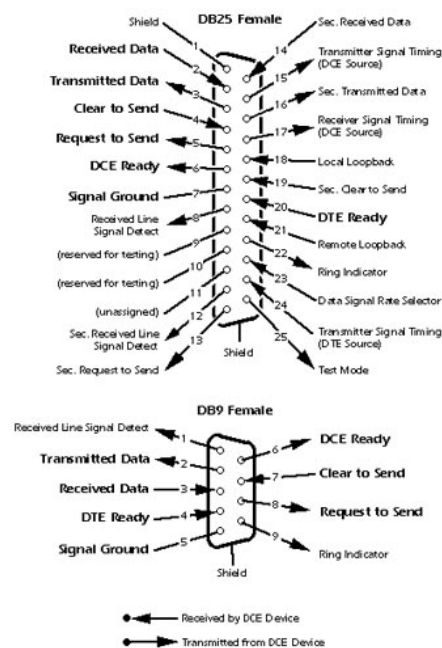


Figura A.4: Conector de um dispositivo DCE

Quando o dispositivo DCE é um modem, para realizar a comunicação muitos sinais são necessários para as conexões. No caso em que o dispositivo DCE não é um modem, ou quando dois dispositivos DTE são ligados diretamente poucos sinais são necessários.

É de se notar, que nas figuras apresentadas existe um segundo canal que inclui um conjunto de sinais de controlo duplicados. Este canal fornece sinais de gestão do modem remoto, fazendo com que seja possível a mudança da taxa de transmissão durante a comunicação, efetuando um pedido de retransmissão se erros forem detetados.

De seguida será apresentado a função dos pinos mais usados na comunicação série:

- Pino 1 - GND (Protetive Ground) - Sinal de terra de proteção;
- Pino 7 - SG (Signal Ground) - Sinal terra usado como referência para outros sinais;
- Pino 2 - TD (Transmitted Data) - Este sinal está ativo quando são enviados dados do DTE para o DCE;
- Pino 3 - RD (Received Data) - Está ativo quando o DTE recebe dados do modem ou DCE;
- Pino 4 - RTS (Request to Send) - Este sinal é um pedido para enviar dados a partir de um DTE. O dispositivo espera até que o CTS fique ativo;
- Pino 5 - CTS (Clear to Send) - O CTS é a resposta do DCE que informa o DTE que este pode transmitir dados;

- Pino 6 - DSR (Data Set Ready) - É um sinal do DCE que indica que o dispositivo está ligado e pronto a receber dados;
- Pino 20 - DTR (Data Terminal Ready) - Funciona da mesma maneira que o DSR mas este sinal vem do DTE;
- Pino 8 - DCD (Data Carrier Detect) - Sinal de saída de um DCE que indica que existe um *carrier* entre os DCE e indica também que a conexão está pronta para a comunicação;
- Pino 24 - EC (External Clock) - É um sinal usado na transmissão síncrona quando é necessário *clock data*. Este sinal é uma entrada no DCE;
- Pino 15 - TC (Transmit Clock) - Transmite o *clock* do DCE em sistemas síncronos;
- Pino 17 - RC (Receive Clock) - Sinal que indica o *clock* recebido no DTE para a decodificação dos dados;
- Pino 22 - RI (Ring Indicator) - Sinal de saída de um modem que indica que foi recebido um sinal de toque.

Referências

- [1] Nanotec Electronic. Pd4-n technical manual v1.5, Junho 2013.
- [2] K.H. John e M. Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-making Tools*. Springer, 2010.
- [3] Flavio Bonfatti, Paola Daniela Monari, e Umberto Sampieri. *IEC 61131-3 programming methodology: software engineering methods for industrial automated systems*. ICS Triplex ISaGRAF, 2003.
- [4] Lolitech. Beremiz user manual, 2008. URL: <http://www.beremiz.org>.
- [5] PLCOpen Technical Committee 2 Task Force. Function blocks for motion control version 2.0, March 2011.
- [6] Slides da unidade curricular arquiteturas de comunicação industrial, 2013.
- [7] Jerzy Kasprzyk. Lecture: Iec 61131-3: Programming languages, Maio 2001.
- [8] C. Neves e L.Duarte e N. Viana e V. Ferreira. Os dez maiores desafios da automação industrial: as perspectivas para o futuro. *II Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica João Pessoa*, 2007.
- [9] A.A.B. Buccioli e E.R. Zorzal e C. Kirner. Usando a realidade virtual e aumentada na visualização da simulação de sistemas de automação industrial. *SVR2006-VIII Symposium on Virtual Reality*, 2006.
- [10] Edmur Canzian. Comunicação série. URL: <http://www.professores.aedb.br/arlei/AEDB/Arquivos/rs232.pdf>.
- [11] Nanotec Electronic. Programming manual v2.7, June 2013.